2

GL-TR-90-0309

DESIGN OF AN ENHANCED GLOBAL SPECTRAL MODEL

T. Nehrkorn, R.N. Hoffman, and J.-F. Louis

Atmospheric and Environmental Research, Inc.
840 Memorial Drive
Cambridge, MA 02139-3794

October 26, 1990

Scientific Report No. 1

Approved for public release; distribution unlimited

GEOPHYSICS LABORATORY
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
HANSCOM AFB, MASSACHUSETTS 01731-5000

"This technical report has been reviewed and is approved for publication"

Donald C. Norquist
DONALD C. NORQUIST
Contract Manager

Donald A. Chisholm
DONALD A. CHISHOLM, Chief
Atmospheric Prediction Branch

FOR THE COMMANDER

Robert A. McClatchey
ROBERT A. McCLATCHEY, Director
Atmospheric Sciences Division

This report has been reviewed by the ESD Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS).

Qualified requestors may obtain additional copies from the Defense Technical Information Center. All others should apply to the National Technical Information Service.

If your address has changed, or if you wish to be removed from the mailing list, or if the addressee is no longer employed by your organization, please notify GL/IMA, Hanscom AFB MA 01731-5000. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
October 26, 1990

3. REPORT TYPE AND DATES COVERED
Scientific No. 1

4. TITLE AND SUBTITLE
Design of an Enhanced Global Spectral Model

5. FUNDING NUMBERS
PE 63707F
PR 2688 TA 04 WU JB
Contract F19628-89-C-0112

6. AUTHOR(S)
T. Nehrkorn
R.N. Hoffman
J-F. Louis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Atmospheric & Environmental Research Inc.
840 Memorial Drive
Cambridge, MA 02139-3794

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Geophysics Laboratory
Hanscom AFB, MA 01731-5000

Contract Manager: Donald Norquist/LYP

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
GL-TR-90-0309

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objective of this study is the development of a vectorized, multiprocessing global spectral model (GSM) with enhanced physical parameterizations. The starting point for this work is the current Geophysics Laboratory (GL) GSM. We have described our multitasking and vectorization design for the GSM. We propose to implement the latitude tasking scheme for multiprocessing the loop over latitude in the calculation of spectral tendencies and adjusted model variables. The general truncation version of the hydrodynamics code will be used. Wavenumber calculations will be multiprocessed and vectorized over wavenumber. All gridpoint calculations will be vectorized over longitude. The physics packages will be modified to bring them into close compliance with the "plug compatibility" rules.

14. SUBJECT TERMS
Numerical weather prediction, multiprocessing, vectorization, global spectral model

15. NUMBER OF PAGES
58

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
SAR

Contents

List of Tables

List of Figures

1 Introduction

The objective of this study is the development of a vectorized, multiprocessing global spectral model (GSM) with enhanced physical parameterizations. The starting point for this work is the current Geophysics Laboratory (GL) GSM.

The global spectral model developed by GL has evolved from the GSM used at NMC as described in (Sela, 1980). The hydrodynamics were completely redesigned (Brenner *et al.*, 1982; Brenner *et al.*, 1984). This model has been used by the Air Force Global Weather Central (AFGWC) with the physics routines taken almost intact from NMC (circa 1983). More recently, an enhanced suite of physical parameterizations has been developed by different investigators and integrated by GL personnel. The new physics routines consist of a coupled boundary layer and soil model (Mahrt *et al.*, 1984; Mahrt *et al.*, 1987), a radiation parameterization (Liou *et al.*, 1984; Ou and Liou, 1988), and a modified Kuo convection parameterization (Soong *et al.*, 1985). The new physics routines have been used in a research mode in a number of studies by GL personnel which demonstrated their potential for improving forecast skill.

As currently implemented, the new physics package requires significantly more CPU time than the baseline GWC physics. Thus, before the potentially better forecast skill can be realized in the operational environment, the execution time of the enhanced global spectral model must be reduced.

Implementation of the code on the Cray 2 offers two major avenues for reducing wall-clock execution time: vectorization and parallelism. Vectorization reduces the CPU time (and, hence, the wall-clock time) spent by a single processor by performing identical (or similar) operations on a number of storage locations in a pipeline fashion. This technique was pioneered by Cray Research, and extensive support tools for vectorizing scalar codes are now available as part of compilers on Cray and other vector machines. To take full advantage of the potential speedups, however. it is necessary to take into account vectorization considerations during the design phase of program development.

A more recent trend in computer technology is the construction of machines composed of a number of processors linked together, allowing multiple simultaneous computations. Such multiprocessing computers contain two or more, in some cases tens of thousands, of individual processing units. These processors may operate in lock step or they may be nearly autonomous. The former case, single instruction multiple data (SIMD), is particularly relevant for image processing applications, while the latter case, multiple instruction multiple data (MIMD), is of more general interest and includes the Cray 2 and Y-MP. Multiprocessor computers give a substantial increase in computing speed for large complex numerical problems. Thus, they provide an opportunity to reduce significantly the computing time for GCM studies and NWP. However, a special effort is required to exploit this opportunity. According to the recommendations of GARP Special Report No. 43 ((WMO, 1984)), "[I]ncreasing computer power and changes in computer architecture...[have] always played a major part in determining integration techniques and will continue to do so if advances in computer technology are to be fully exploited. Therefore, study to determine the most efficient or appropriate algorithms will continue to be necessary, taking into account, for example, the future use of multiprocessors with a very large storage.

Development costs of such optimized computer codes will be high, and they should be designed from the outset to be widely usable..."

In this report, we present the results of our analysis of the existing GL GSM code, and develop a comprehensive design for vectorizing and multiprocessing the GL model. Our changes to the existing code are guided by two general strategies.

Enhance clarity. Clear, direct coding is best. Coding which makes special use of a particular machine architecture and compiler is in the long run counterproductive. Eventually, and probably sooner than later, the compiler will be updated or the code will be transported to other machines. Generally, as time goes on, compilers get more and more capable and what seemed to be necessary last year is now a hindrance. Complicated coding stategies to squeeze the maximum horsepower out of the CRAY2 will be restricted to very specific procedures and isolated in individual procedures. For the most part these procedures will be standard library routines for which vanilla versions are readily available. An example is the FFT software.

Avoid impacts. Do not make changes which will have a big impact on the results. Changes which result in small changes in the tendencies, which are not less correct are fine. Some differences in the calculated tendencies are unavoidable. It is expected that different versions of the model, or the same version compiled differently will usually result in a different order of arithmetic and hence in different round off errors. On the other hand changes which result in large differences in the tendencies, even though they may be as correct as the original, should be avoided. For example, one could argue that a speedup is possible by using a T150 truncation instead of a R120 truncation. However, demonstrating that this is indeed the case would require a substantial suite of test cases, well beyond the scope of the present effort.

In the background section, we review the basic properties of the GSM algorithm, and discuss the implications for multiprocessing and vectorization. Our design criteria are discussed in detail in section 3. In section 4 we describe and discuss the current GL GSM, and go on to describe the changes necessary for our proposed design. The final section contains a summary and suggested further improvements and developments that are beyond the scope of the present effort.

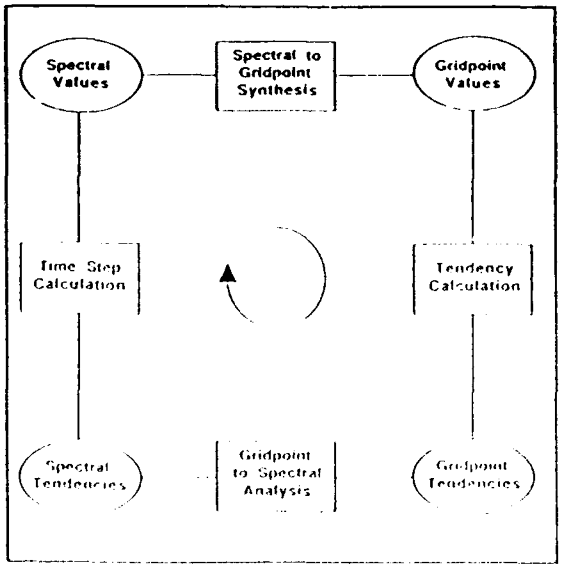
2 Background

Before discussing the details of the GL GSM design, it is useful to consider the basic characteristics of a GSM in general. In the next section, we present a highly simplified view of the basic GSM algorithm, which nonetheless contains all the important elements one needs to consider for optimizing the code. Following this discussion, we present two alternative approaches to multitasking the GSM, and discuss issues of vectorization and storage allocation. It is important to note that a significant portion of our optimization effort will be directed at vectorizing the gridpoint code, in particular the physics calculations, which are unaffected by the choice of the multitasking scheme.

2.1 Parallelisms in basic GSM algorithm

The GSM actually does most calculations in gridpoint space. Only linear terms are calculated in spectral space. Other terms, such as advection, diabatic physical processes such as moist convection, etc. are localized in grid point space and much easier to calculate there. If we consider, for the sake of explanation, an adiabatic spectral model, then a simple graphical description of the functioning of the model is given in Fig. 1. Beginning with spectral values of the variables (upper left corner), the model transforms these to values of the grid point variables and their spatial derivatives. These are used to calculate the nonlinear part of the tendencies which are then transformed back to spectral space. Some linear terms are added to the tendencies in spectral space, including higher order horizontal diffusion. The tendencies in spectral space and variables at previous time levels in spectral space are then used by the time stepping procedure to obtain spectral values at the new time. The cycle is continued until the desired forecast length is achieved.

Figure 1 Functional overview of GSM algorithm



As the basis for describing our multiprocessing algorithms, we first consider a representative but schematic GSM algorithm. Let $X(\lambda, \mu)$ be all gridpoint variables in all layers at longitude λ and sine of latitude μ. The Fourier transform of X is X_m and the Legendre transform of X_m is X_n^m where m is the longitudinal wavenumber and n is the meridional mode number. For clarity of notation, μ, λ and normalization factors will be omitted in most of the equations that follow.

A representative time step is composed of the following steps:

(1) Legendre synthesis at each μ for each m:

$$X_m = \sum_n X_n^m P_n^m, \tag{1}$$

where P_n^m are the associated Legendre polynomials. Meridional derivatives of the fields are also calculated at this time by replacing P_n^m by its derivative in the above expression.

(2) Fourier synthesis at each μ for each λ:

$$X = \sum_m X_m e^{im\lambda}. \tag{2}$$

(3) Gridpoint operations at each (λ, μ) coordinate:

$$f = F(X) \tag{3}$$

where $f(\lambda, \mu)$ is a vector of gridpoint tendency and flux terms in all layers, and where F is a nonlinear function of X and its derivatives at a single latitude-longitude point. This step involves vertical coupling.

(4) Fourier analysis at each μ for each m:

$$f_m = \sum_\lambda f(\lambda)\, e^{-im\lambda} \tag{4}$$

(5) Gauss-Legendre integration for each m and n:

$$f_n^m = \sum_\mu w(\mu)\, P_n^m(\mu)\, f_m(\mu) \tag{5}$$

where $w(\mu)$ are the Gauss-Legendre weights. Flux divergence terms are integrated by parts. A more complicated, but for present purposes equivalent, expression is used for these terms.

(6) Time integration for each m and n. Once the time tendencies are assembled, the model state is advanced in time by solving the semi-implicit equation, which is schematically given by

$$\begin{gathered}\left(X_n^m|_{t+\Delta t} - X_n^m|_{t-\Delta t}\right)/(2\Delta t) = \\ (f_n^m - L_n X_n^m)|_t + L_n(X_n^m|_{t+\Delta t} + X_n^m|_{t-\Delta t})/2\end{gathered} \tag{6}$$

where L_n is the linear operator governing gravity wave evolution and all quantities are evaluated at the time levels indicated. The operator L_n couples vertical layers. At the completion of step (6), we are ready to begin step (1) again.

The dependencies and parallelisms in each of these steps are summarized in table 1. In this context "dependency" means that a value of an array at index l depends on values of that same array at other values of l, and "parallelism" means that operations can be performed on several array elements simultaneously. The conventional implementation of this algorithm is to initialize the spectral tendencies to zero or to their linear components and then perform steps (1) through (5) with a loop on latitude, accumulating the tendencies in f_n^m. Except for the accumulation in step (5), the computation for each latitude is independent. This implementation makes efficient use of storage, since oniy the gridpoint variables for a single latitude are needed at any one time. (Actually, two latitudes at a time are usually processed, one in each hemisphere; this allows further efficiencies by making use of symmetry properties.)

Table 1 Analysis of dependencies and parallelism in the basic GSM algorithm. The letters in the last two columns denote: i-longitude, j-latitude, k-vertical layer/level, m-zonal wavenumber, n-meridional mode number

Step No	Description	Dependencies	Parallelisms
1	Legendre synthesis	n	j,k,m
2	Fourier synthesis	i,m	j,k
3	Gridpoint calculations	k	i,j
4	Fourier analysis	i,m	j,k
5	Gauss-Leg. integration	j	k,m,n
6	Time integration	k	m,n

All physical processes might be included in the tendency calculation. However, and this has generally been the case, if some are adjustment processes, such as a dry adiabatic adjustment or a moist convective adjustment, then a second set of transforms is required. This second loop is like the first (see Fig. 1) except instead of calculating tendencies we calculate adjusted values and instead of time stepping we replace values. In this diabatic model the two loops alternate.

In the calculation of the vertical diffusion in the PBL, the time stepping is done implicitly in grid-point space, then tendencies are re-calculated and added to the tendencies of the other process.

There are two favored truncations in spectral space, rhomboidal and triangular. The names rhomboidal and triangular describe the shape of the region in the m,n plane retained in the truncation. Since the meteorological fields are real valued only non negative values of m are retained. For both truncations m varies from 0 to N_m. For rhomboidal truncation n varies from m to $m+N_n$, while for triangular truncation n varies from m to N_n.

2.2 Multiprocessing strategies

In the next two sections we introduce two different schemes for multitasking the basic GSM algorithm. We will implement the first one of these (latitude tasking), because it is more straightforward to implement, without any significant performance penalty for the envisioned target machines. The choice of multiprocessing scheme only affects the wavenumber calculations (step 6) and the transform steps (steps 1, 2, 4, and 5): the gridpoint calculations (step 3) are treated in the same way in both schemes. Thus, while changing the multiprocessing scheme would involve substantial changes to the hydrodynamics, the physics packages would be unaffected.

In the following we concentrate exclusively on multiprocessing the computations. Generally, we will avoid I/O within the computations. Diagnostic output should be saved in a global data structure and written outside of the multiprocessed sections. However, for radiative heating rates, we must store results between time steps and between latitudes. Currently, the radiation code couples latitudes pairwise. That is, the results from latitude *n* are saved and applied to latitude *n*+1. For the transfer between latitudes to insure that the heating rates from latitude *n* are available to latitude *n+1*, the latitudes must be allocated in pairs to the processors. This can be accomplished explicitly or by making use of optional parameters on the compiler directives. The heating rates may be read from the data set just written or simply saved between subroutine calls for two adjacent latitudes. The easiest solution between time steps is to use a direct access data set and read the values as needed. This requires that the operating system keep track of multiple (perhaps simultaneous) read requests to a single data set. Using multiple files would be much less desirable.

Latitude tasking scheme

The first approach, denoted latitude tasking, is the most straightforward adaptation of the GSM to a *multiprocessing* environment. In the latitude macrotasking scheme, the computations of step (1) through (5) are considered one task, which is handled by different processors for different latitudes. The time integration, step (6), is considered a task to be handled by different processors for different spherical harmonics or modes.

In the latitude tasking scheme, all variables are shared among the processors except for those that are defined at a single latitude. Thus only the values of the Legendre polynomials, the Fourier coefficients and gridpoint values of the variables, and the gridpoint values of the nonlinear terms are local to the processors. Within the loop over latitude, all the shared variables are read-only, with the exception of the spectral coefficients of the tendencies. Thus, the integrity of the data is readily preserved within the latitude loop, as long as the updating of the tendencies in step (5) is safeguarded. During the time-stepping, the computations in step (6) for one mode do not affect those of any other mode, thus eliminating any need for safeguarding shared data before it is updated.

Latitude wavenumber tasking scheme

For latitude wavenumber macrotasking, we consider the calculations in steps (2), (3) and (4) for each latitude *j*, to be a task handled by one of the processors. Similarly, we consider the calculations in steps (5), (6) and (1) for each wavenumber *m*, to be a task. This implies that steps (1)-(5) are no longer performed in a loop on latitude as in the conventional single processor implementation. It is possible to go further and consider each of steps (1) through (6) to be a separate task, but this adds synchronization points and storage requirements.

Within each task, a processor can proceed independently of the others: no communication is necessary and, hence, the integrity of the data is readily preserved. Since no partial sums are used, the ordering of tasks is immaterial. Within tasks, arithmetic will follow prescheduled ordering so that results are exactly reproducible. Between tasks the ordering of data movement is immaterial. These transitions (1)-(2) and (4)-(5) require processors to share the results of their computations, and care is required for them to do this efficiently an without compromising the data. Essentially a processor's results from step (1) are placed in memory for all processors to read when performing step (2). The same procedure is followed for steps (4)-(5). One method of insuring data integrity is to maintain a critical set of indicators of the valid time of each segment of shared memory. An alternative is to introduce synchronization points at the end of the latitude and wavenumber tasks.

Storage requirements for this scheme are not much greater than other schemes. In the latitude tasking scheme, we maintain three complete sets of spectral coefficients, for the previous two time levels and to accumulate the spectral tendencies. In the current scheme, the spectral tendencies are accumulated locally, for each wavenumber, so we need to keep only two complete time levels of spectral data. However we also need to store the Fourier coefficients for the variables and tendencies. If the Fourier variables and tendency data are equivalent and only the minimal truncation is kept, then the Fourier data require $2*N_j/N_n$ times the storage of one set of spectral coefficients.

By breaking the basic algorithm at the Fourier representation we obtain macrotasking for both latitude and wavenumber tasks. Here the latitude tasks are somewhat smaller than those used in the latitude only macrotasking scheme. The advantages of latitude wavenumber tasking are that the entire algorithm is macrotasked, the Legendre calculations are completely accomplished within a single task so that reproducibility is assured.

2.3 Vectorization strategies

For vectorization, only the innermost loop is considered. Hence we will rearrange the code and in some cases the algorithm so that the innermost loop is on latitude or longitude or wavenumber. In some cases two small innermost loops can be combined into one large one. In other cases small innermost loops can be unrolled.

2.4 Storage strategies

There has been some discussion as to whether complex storage is inefficient on the Cray, because referencing the real parts of a vector requires a stride of two. In the GSM most of the complex arithmetic is of the form of a real matrix times a complex vector. This takes place both in the Legendre transforms and the time stepping. We have conducted some experiments, timing various approaches to the Legendre synthesis. The results are so mixed that it seems unwise to base our storage strategy on this consideration. Furthermore, breaking complex numbers into real and imaginary parts is counter to our general strategy to enhance clarity.

Stack allocation provides automatic workspace. It is required for multitasking anyway. This is CRAY2 specific, but does result in clear coding. Temporary arrays in the version developed will include all variables not declared in the main routine or in common blocks. When migrated to another architecture, all variables declared in subprograms or stored in task common blocks are temporaries and could be allocated by a work space strategy.

3 Design Criteria

The overall design of the GSM is influenced by the target architecture. Our design is geared toward a coarse-grained, shared memory multiprocessor with vector processors, such as the Cray 2 or the Cray Y-MP. Coarse-grained in our context means that there should be no more processors than there are latitudes in one hemisphere. For current and proposed supercomputers like the Cray 2 and its successor, this condition is easily met, especially if the spatial resolution of the model is ·ncreased to take advantage of the increased computational speed.

Aside from this basic design constraint, there are a number of other issues that will be considered and incorporated into our design. The Statement of Work of this contract identified a number of requirements that the design must meet. Another source for design and coding guidelines are the "plug compatibility" rules designed to allow easy interchange of physics packages (Kalnay *et al.*, 1989). Not all of these rules will be incorporated into the present design, but we will identify our violations of these rules and indicate how to remedy them in future upgrades to the code. We will discuss both sets of constraints in the following, grouped into the following categories: (1) Efficiency, (2) Data integrity and reproducibility, (3) Flexibility, and (4) Coding style.

3.1 Efficiency

For maximum efficiency, both the multiprocessing and vectorization capabilities of the target machine must be exploited as much as possible, and in such a way that they complement rather than compete with each other. For maximum multiprocessing efficiency, the sequential portion of the code must be minimized. Thus, as many of the computations as possible must be performed in parallel. This will be accomplished in our design by multitasking the latitude parts of the computations, and microtasking the wavenumber tasks. Other sources of multiprocessing inefficiency are load imbalance, memory contention, and overhead. In general, static task allocation requires less overhead than dynamic scheduling, but it has the potential for greater load imbalance. Our choice of task allocation mechanism will be determined by which of these considerations is more important for the latitude tasks. The most important source of memory contention is the requirement for two processors to lock and write to the same memory location. Our design will avoid writes to the same memory location by different processors in all but a few, well isolated sections of the code.

For maximum vectorization efficiency, the innermost loops must have many iterations, and the code inside the loops must avoid structures (such as complicated if-constructs and iterative or recursive code) that inhibit vectorization. In addition, to avoid memory conflicts, it is best to have unit (or at least odd) strides in the innermost loop. In the design of the physics packages (and all other gridpoint calculations), we incorporate these constraints by choosing the innermost loop to be over longitude, and arranging the data storage such that the innermost index is also over longitude.

Aside from vectorization and multiprocessing, scalar optimization of the code can lead to further speedups. In particular, use of standardized software where possible will allow the use of specialized libraries that are optimized for the machine on which the GSM is installed.

3.2 Data integrity and reproducibility

A primary concern with any multiprocessing design is the preservation of data integrity: different processors must not be allowed to simultaneously read from and write to a particular variable. This requires that different processors use memory shared by all processors as read-only, and only write to memory locations reserved for them; writes to global memory must be safeguarded with software locks. In our design it is easy to identify read-only and read/write variables in the hydrodynamics, and the physics packages will be modified to allow a clear identification of input and output variables. In particular, initialization of variables currently done in the first call to a routine will be moved to a separate initialization routine, which will be called in the sequential start-up portion of the code.

A related, but separate question is the issue of reproducibility. In the context of the software design, this refers to the ability to exactly reproduce results in subsequent runs of the same code, using identical data. For exact reproducibility, all arithmetic operations must be performed in the same order. Thus, where sums are accumulated across processors as in our latitude tasking scheme, the order in which the tasks are completed by the different processors must be predetermined and held fixed. We will include an option to enforce this with software locks in our design; for operational applications, where reproducibility is unimportant, this option will be disabled and the potential overhead cost due to processor idle time will be avoided.

3.3 Flexibility

In order for the code to remain a useful research and forecast tool, it must be flexible enough to accommodate new developments in algorithm development and machine architecture.

To allow easy migration to different computers, the machine specific aspects of the code (multitasking and software lock calls and directives, designation of memory as global or local to each processor) will be isolated as much as possible so that only a small portion of the code has to be changed. The remainder of the code will adhere to standard Fortran 77. We plan to implement multiprocessing by using microtasking compiler directives. These directives activate system routines which use the number of processors currently available.

The code will be written to support variable vertical and spectral resolutions. The hydrodynamics will support any pentagonal truncation (the currently used rhomboidal and the frequently used triangular truncations are both special cases of a pentagonal truncation). In addition, computations may be pe^formed over only one hemisphere and/or over only a section of the globe bounded by meridians.

Finally, to allow easy interchange of physical parameterizations, the "compatibility rules" must be adhered to as much as possible. Some of these rules are intended to allow vectorization and multitasking, and to promote good coding style and are thus discussed in those sections. The following rules are particularly geared for variable resolutions and will be adhered to in our physics packages: the vertical dimension is variable, and all dimensions of arguments are contained in the argument list. Local work space will be dimensioned by PARAMETER statements within the physics subroutines.

3.4 Coding style

There are a number of stylistic guidelines that will all be adhered to in the hydrodynamic part of the calculations, and that will be incorporated as much as possible into the physics packages.

The structure of the code will be top-down and modular. Thus, the program execution generally proceeds from the beginning to the end of a module, and logically distinct tasks are separated into different modules. This implies that obsolete coding practices (arithmetic if-statements, gotos, etc.) will be eliminated in favor of structured code.

Internal documentation will be used to explain and document algorithms and program flow, inputs and outputs to routines, and to identify machine-dependent sections of the code.

The physics packages will adhere to a number of stylistic compatibility rules. There will be only two entry points at the interface between the package and the rest of the model: one for initialization of variables and one for running. The packages will not use any blank commons. No array indices will exceed their declared dimensions.

The remaining compatibility rules will not be completely met. In particular, communication between the package and the model will not only be through the argument list. Instead, where commons are used to pass values, they will be clearly identified. The packages will also be allowed to use external references other than their own routines, but again these will be identified. The packages will also continue to use STOP statements to terminate execution in the case of error conditions. These will be limited to a small number, however. Finally, we will continue to allow I/O that cannot be turned off; this is necessary for the radiation code. However, we will require that all I/O is only to units that are either passed in via the argument list or set in PARAMETER statements, and no I/O to standard output will be allowed.

4 Current GSM and proposed design

The proposed design for the GSM is presented in this section. For each part of the code, we first discuss the current GSM, and proceed to detail the needed changes for implementing our design.

4.1 Basic approach and hydrodynamics

The current GSM follows the conventional implementation of the basic GSM algorithm discussed in section 2.1: all gridpoint calculations are performed in a loop over latitude, with partial sums of the spectral tendencies (or adjusted values, in the case of the adjustment physics) accumulated during the loop. Currently, most gridpoint calculations are performed one vertical column at a time, with longitude being the outer dimension. Only rhomboidal truncations for global simulations are supported. The spectral and vertical resolution can be varied; the corresponding array dimensions are given as edit symbols in the code, which need to be replaced by a batch editor prior to compilation.

As a first step in identifying the most promising areas of improvement, and to provide a baseline for future comparisons, we have conducted several short (1-hour) test forecasts on the Cray 2, using different levels of compiler optimization. The timing results from these test runs are summarized in table 2. These test runs all used a 20 minute time step, with radiation calculations only performed every 3 hours (i.e., only at the first time step for each run). A rhomboidal 30 truncation with 18 layers in the vertical was used. The timings in table 2 are single processor CPU times (in sec) for one time step. Results are shown for a time step with and without radiation, and for the average (assuming 1 time step with radiation every 9 time steps). It is obvious that the radiation calculation dominates CPU usage for time steps with radiation, even though it is only performed every other latitude. Of the remaining calculations, the most expensive are (in order) the transforms, the boundary layer calculations, the moist physics (which also performs transforms), and time stepping.

Comparing timings for the runs without any optimization, and with scalar compiler optimization, shows speedups by a factor of approximately 4 for most modules, with the notable exceptions of the radiation (2.3) and time stepping (2.8). Speedups due to vectorization are smaller for all modules. For purposes of illustration, the vectorized percentage of the utilized code has been computed from these speedups using Amdahl's law, assuming a ratio of 10 for the execution speed of a vectorized over a scalar instruction (a number typical for Cray-XMPs (Levesque and Williamson, 1989)). Of the CPU-intensive modules, the transforms have the highest degree of vectorization, followed by the moist physics. Time stepping and adiabatic nonlinear products show notably smaller speedups, and the boundary layer physics and the radiation have very small speedups.

The poor vectorization of the physics routines was to be expected, since all calculations are performed in vertical columns, resulting in short inner loops. (Vectorization of short loops results in a relatively large loop overhead with correspondingly small increases in computation speed.) Of all the physics routines, the radiation code is the most CPU-intensive, even for the "average"

Table 2 Timing statistics for one time step of the GSM, for three levels of compiler optimization (None, Scalar, Vectorized). See text for details.

		NONE	SCALAR		VECTORIZED					
						Percent of Time Step				
	Program Step	Time	Time	Speedup	Time	w/rad	w/o rad	avg.	Speedup	Percent Vectorized
1.	Time Step									
	w/rad	248.68	105.92	2.3	71.39	100%			1.5	37%
	w/o rad	98.85	24.72	4.0	11.25		100%		2.2	61%
	avg.				17.93			100%		
1.1	Laloop									
	w/rad	220.42	97.70	2.3	67.98	95%			1.4	32%
	w/o rad	70.31	16.57	4.2	7.42		66%		2.2	61%
	avg.				14.15			79%		
1.1	Subrad									
	w/rad	151.83	81.32	1.9	60.63	85%			1.3	26%
	w/o rad	.24	.07	3.4	.07		0%		1.0	0%
	avg.				6.80			38%		
1.2	Blflux	19.00	4.56	4.2	3.04	4%	27%	17%	1.5	37%
1.3	Nlprod (adiab.)	5.13	1.18	4.3	0.68	1%	6%	4%	1.7	46%
1.4	Transforms	46.02	10.73	4.3	3.69	5%	33%	21%	2.9	73%
2.	Diffsn	.08	.02	4.4	0.01	0%	0%	0%	2.6	68%
3.	Implct	8.36	2.79	2.8	1.61	2%	14%	9%	1.8	40%
4.	Conpcp	19.89	5.35	3.7	2.21	3%	20%	12%	2.4	65%

time step, and the most poorly optimized, and thus the prime target for improvements. It is clear, however, that there is room for improvement in the other modules, as well. Moreover, the other physics routines, the transforms, the time stepping, and the adiabatic nonlinear products all take up a nonnegligible part of the time step, and thus all require at least some improvements.

Figure 2 Flowchart for the main routine and STEP1 and STEPN. Not shown are utility routines, and call trees for the physics packages (these are shown separately in subsequent figures). A nonzero value for IDBUG will terminate the forecast after the first unfiltered step; for IDBUG > 0, various calculations inside STEP1 will be skipped

```
START  -

  (A)  Initializations
       - BSCST     (basic constants)
       - GAUSLT - GETEPS  (compute Gaussian latitudes)
                - POLY
       - MCOEFF - IMINV   (precompute matrices for time step)
                - MULTMX
       - INIPMN           (precompute constants for Pmns)
       - INIFFT           (initialize FFTs)
       - INIRAD           (precompute constants for radiation)
       - INIPBL           (precompute constants for PBL)
       - INIEST           (precompute constants for sat. vapor pressure)
       - INIADJ           (precompute constants for adjustment physics)

  (B)  Input initial data
       - Restart:
           - RHIST   - RDATB    (read history data set)
                     - RDATC
                     - RDSFC
           - goto (D)
       - Initial Start:
           - RDATA   - RDATB    (read initial data set, compute stats)
                     - RDATC
                     - RMSCAL - RMS2
                     - GWMASS - PMNS
                              - LSUMB,LSUMC,SATONS
                              - FFTN
                     - RDSFC

  (C)  Initial time step
       - STEP1  -
  - LALOOP      (latitude loop - see below for details)
  - if idbug=3 goto (C1)
  - DIFFSN      (horizontal diffusion)
  - if idbug=2 goto (C1)
  - IMPLCT      (time stepping)
              (C1) Copy time level 3 to time level 2 for Z,D,Q
                - if idbug > 0 goto (C2)
                - CONPCP       (adjustment physics)
              (C2) Copy time level 3 to time level 2 for T,W
       - if IDBUG.ne.0 then
          - WNMI - WDATB       (write out tendencies or preliminary
                 - WDATC        values, in format compatible with NMI)
          - goto (F)
       - if initial data output specified:
       - WDATA - WDATB         (write out initial data set, compute stats)
               - WDATC
```

```
             - RMSCAL - RMS2
             - GWMASS - PMNS
                      - LSUMB,LSUMC,SATONS
                      - FFTN

(D)  Beginning of time step loop
     - STEPN  -
              - LALOOP  (latitude loop - see below for details)
              - DIFFSN  (horizontal diffusion)
              - IMPLCT  (time stepping)
              - TFILT   (time filter for Z, D, Q)
              - CONPCP  (adjustment physics)
              - TFILT   (time filter for T, W)
     - increment time counters
     - if data is to be written out at current step:
     - WDATA  - WDATB   (write out forecast data set, compute stats)
              - WDATC
              - RMSCAL - RMS2
              - GWMASS - PMNS
                       - LSUMB,LSUMC,SATONS
                       - FFTN
     - if restart data is to be written out at current step:
     - WHIST  - WDATB   (write out restart data set)
              - WDATC
              - WRTSFC
     - if forecast not finished, goto (D)

(E)  End of forecast
     - output data and history data set if not already done

(F)  End of program
     - stop
```

We propose to implement the latitude tasking scheme described in section 2.2. At the same time, we will implement the general truncation version of the hydrodynamics code developed under a previous contract. This version of the code is already set up to clearly separate storage that is shared among the processors (global storage) from storage local to each task ("task common" in Cray terminology). The flow chart for this version of the hydrodynamics of the GSM is shown in Figs. 2, 3.

Figure 3 Flow chart for the LALOOP subroutine.

```
START  -
```

```
(A) Begin loop over latitude pairs
     - PMNS (compute Legendre polynomials for current latitude pair)
     - SPTOGP (spectral to gridpoint transformation)
               - LSUMB, LSUMC  (Legendre sums)
               - UMSVMA, UMAVMS (computation of winds)
                        - LSUMB, LSUMC
               - SATONS (conversion of symmetric/antisymmetric to NH/SH)
               - FFTN (FFTs)
     - NLPROD (nonlinear products)
     - SUBRAD (radiative heating rates)
     - BLFLUX (PBL physics)
     - GPTOSP (gridpoint to spectral transformation)
               - FFTN   (FFTs)
               - NSTOSA (reverse of SATONS)
               - GQB, GQC (Gauss-Legendre sums)
               - ADVB, ADVC (Gauss-Legendre sums for flux terms)

(B) End loop over latitude pairs

(C) Return
```

We note that the microtasking approach uses dynamic task allocation in the latitude loop. This is an advantage because of the disparate number of physical computations at the different latitudes. Before implementation, we will determine the load imbalance between the latitude tasks (once the vectorization is completed) and determine the best order for the latitude tasks.

The general truncation version (documented in (Kaplan *et al.*, 1985)) supports any pentagonal truncation, including the frequently used rhomboidal and triangular truncations. In addition, computations may be performed over only one hemisphere or a section of the globe. Array dimensions are given in terms of PARAMETER statements. The parameter statements and the common blocks are stored in include files, which are incorporated into the code at compile time. Edit symbols are retained, however, to allow to choose between the hemispheric and global versions of the code, and to use double precision for certain, sensitive calculations (useful for use of the code on 32-bit machines; see (Nehrkorn, 1990)). This version of the code also makes use of Hoffman's (Hoffman, 1985) semi-implicit time stepping scheme, which uses full temperature (as opposed to perturbation temperature) throughout. This does not preclude introducing a basic state and perturbation temperature for the integration of the hydrostatic equation. In general, the basic state temperature used for the hydrostatic equation and that used for the semi-implicit time scheme will be different. Therefore it is best to use full temperature throughout and introduce linearizations where needed. In fact, this approach is simpler in the cases at hand. Since the hydrostatic equation is integrated spectrally, if the basic state temperature is independent of latitude, longitude and time, then only the coefficients of the single (0,0) mode would be effected.

We will use vectorization within the multitasked latitude loop to exploit the remaining parallelism identified in section 2.1. In particular, gridpoint calculations will be reorganized such that the innermost loop is over longitude.

During the transforms, the Legendre transforms will be vectorized over zonal wavenumber *m*. We have tested various implementations of the Legendre transforms, but found no consistent advantage to changing the storage or loop arrangement (also see section 2.4).

In the wavenumber calculations, we will use both microtasking and vectorization over spectral mode number. In parts of the calculation where the vertical layers are not coupled, we may microtask over the vertical layers instead. To avoid memory bank conflicts we will use different parameters for array dimensions and for the range of indices. For example, if the number of longitudes, mp=128, then all corresponding arrays will have a first dimension of mpDIM=129.

4.2 Radiation

Current status

This routine calculates heating rates in the atmosphere and radiative fluxes at the earth's surface. It uses the current temperature, and humidity profiles on the model sigma structure and the pressure, temperature and albedo of the surface. There is no interaction between the radiation routines and the Kuo and large scale precipitation parameterizations: the radiation routines calculate cloudiness from the relative humidity field. The only interaction with the PBL parametrization is at the surface, through the albedo and radiative flux. The heating rates in the layers are calculated as differences between fluxes at the levels. The fluxes in turn require the calculation of atmospheric transmissivities. The principal reference for the procedures used is (Ou and Liou, 1988). The reader interested in more detail than that provided here should consult that reference, especially Appendix B for a description of the program.

The calculation is divided spectrally into two main parts, the thermal (or IR) and solar (or visible). This division is very useful because there is a well defined spectral separation at about 4.5 μm between the emissions from the sun and the earth. In the thermal band the calculation is more complicated because each layer emits. Furthermore, to attain the same level of accuracy, the different active gases - CO_2, H_2O and O_3 - must be handled separately and the effect of pressure and temperature on absorption must be modeled. In all, six bands are used in the IR because the effective absorber amounts for H_2O are different for three spectral regions and there is an overlap between the CO_2 band and one of the H_2O bands. In the IR, clouds are treated as black bodies. The non-blackness of ice clouds is handled as a correction term (Ou and Liou, 1988, Appendix A). The thermal calculation ignores all scattering processes, including reflection from the ground. In the solar calculation, on the other hand these processes are foremost, while the only energy source is the sun, and only a single band is used. The same three active gases are treated, but the modeling of gaseous absorption is fairly simple. Multiple reflections are accounted for by introducing generating functions, G_i. This technique is sometimes called the adding method.

As already noted, the radiation calculation, averaged over all time steps presently accounts for 50% of the calculation (more detailed timings are given below). At a single point, given the proper cloud parameters, the calculation is quite precise (Ou and Liou, 1988). However considering that modeled clouds are not very realistic, that the calculation occurs only every other latitude, every other longitude and every three hours and that the radiative fluxes depend strongly on cloud, we might argue that the current parametrization is severe overkill and that a reasonable strategy would be to replace the entire radiation calculation with a simpler, faster scheme. However we will not pursue such a strategy since it conflicts with our general principle to avoid impacts.

The following points, requiring attention, were noted in our review of the code:

In the IR calculation the vertical indexing is from the surface upwards, while in the solar calculation the vertical indexing is from the top of the atmosphere downwards.

In the radiation calculation, there are many temporary arrays dimensioned 40, when in fact only the first KP+1 locations are referenced. The constants 40 and 41 appear in a number of places within the subscripts of the variable CL9, which is dimensioned 80.

There appear to be some errors in the calculation of the G_i, for the three cloud case. Specifically, multiple reflections are not accounted for in Eqns. 3.28 and 3.29 of (Ou and Liou, 1988) for the upward diffuse component. Additional errors were noted in Eqs. 3.41 and 3.43.

There appear to be arbitrary constants (EKS1 and EKS2) in the formula which interpolates the IR flux within cirrus clouds. According to Ou (pers. comm.) this interpolation is based on the fact that fluxes in the interior of the cirrus cloud should go to zero. The form chosen is the weighted sum of the fluxes at both boundaries decaying exponentially towards the cloud interior. The constants mentioned determine the rate of exponential decay and were tuned so that the interpolated fluxes agree with a more complicated radiative transfer model (Kinne, 1987). It is not clear to us if these constants should be re-tuned if the sigma structure of the model changes.

In the cirrus correction it is assumed that $\tau_c = 1-\epsilon_c$, although the cirrus transmissivity is in fact calculated separately.

Strategy

Our strategy will be to enhance vectorization and to improve efficiency by applying Horner's rule for evaluating polynomials.

In the radiation calculation there are parallelisms with respect to longitude, band (in the IR) and level. All of these parallelisms are broken by the effect of clouds. The parallelism over longitude is otherwise complete with the exception of the basic difference between day and night calculations. Since the parallelism over longitude is strongest and provides the longest vectors, we will vectorize over longitude.

Since everything is vectorizable with respect to longitude except for cloud effects, it makes sense to calculate as much as possible independently of cloud effects. Since clouds appear almost everywhere, we will treat the general cloudy case at all grid points. We note that the proportion of grid points with some cloud in our test case at time 0 is 86%. Most of the clear areas are at particular latitudes, namely the subtropics and Antarctica, but even at these latitudes there is cloud at more than 50% of the grid points. The algorithms outlined in the next sections treat clouds so that most calculations take place in the same manner whether a cloud is present or not.

There do not seem to be other obstacles to vectorization. In the two main subroutines, IRRAD and IRCLR, there are three main uses of if statements. The most important ones have to do with clouds. The most numerous ones have to do with setting lower or upper limits on certain variables, especially absorber amounts. These could be recoded as MINs or MAXs. Finally, some are related to using the same code in subroutine IRCLR for both upward and downward flux calculations. We expect to develop separate code for the upward and downward IR flux calculations. In the curren: code there are many cases of do-loop indices varying from longitude to longitude because of cloud effects. Some of this is unavoidable, but it can be isolated to the calculation of the adjusted IR source functions, as discussed below.

At the start of the contract we were provided with an older "one-decker" version of the radiation code. Since then the newer "three-decker" version has been totally debugged and it was agreed to use the three-decker code in the remainder of this project. The effect of cloud in the IR calculation appears to be simpler in the newer code. The three-decker is the version described in (Ou and Liou, 1988).

We also propose to use Horner's rule for evaluating polynomials. There are several instances of radiative properties being parametrized in terms of fixed low order polynomials. These are all coded very inefficiently now using an explicit evaluation and short loops. We will unroll these loops and apply Horner's rule. For example, the innermost loop of all is the evaluation of a third degree polynomial for log of emmissivity. The original code and the code using Horner's method are:

```
c     The original emmissivity calculation.
      do 4020 kg=1,ntot
         do 4021 nn= nns,nne
            emmi(kg, nn)= 0.
            if (upath (kg, nn) .lt. 1.e-7) go to 4023
            upath (kg, nn) =(alog10 (upath (kg, nn))*2.-aux(kg))/auy(kg)
            do 4013 mm=1,4
 4013          drk(mm)=rj(kg,nn,mm)*upath(kg, nn)**(mm-1)
            emmi(kg, nn)=exp(drk(1)+drk(2)+drk(3)+drk(4))
            go to 4021
 4023       emmi(kg, nn)=exp(rk(kg,nn))*upath(kg, nn)/1.e-7
 4021       continue
 4020    continue

c     Alternate emissivity calculation.
      do 4020 kg=1,ntot
         do 4021 nn= nns,nne
            if (upath (kg, nn) .ge. 1.e-7) then
               upath (kg, nn) =(alog10 (upath (kg, nn))*2.-aux(kg))/auy(kg)
               emmi(kg, nn)=exp(rj(kg, nn,1) + upath(kg, nn)*(
     &              rj(kg, nn,2) + upath(kg, nn)*(
     &              rj(kg, nn,3) + upath(kg, nn)*rj(kg, nn,4))))
            else
               emmi(kg, nn)=exp(rk(kg,nn))*upath(kg, nn)/1.e-7
            endif
 4021    continue
 4020 continue
```

There is a substantial difference in the timings of these two codes. We used FLOWTRACE to time the various parts of the radiation code, using the one-decker version. These parts can be either subroutines or small segments of code. In either case the reported times are exclusive of the times for included sections of code which are also timed. For example, in the results below, since we timed only the radiation routines, the time fc· MAIN includes all calculations except radiation. Since the only change is in the IRCLR/EMIS calculation, the results from the two runs demonstrate that the timings are relatively stable with respect to other activity on the machine.

Figure 4 Timings (from FLOWTRACE) for the original and alternate radiation calculations.

```
                         ORIGINAL                 ALTERNATE
Routine                 Time executing           Time executing    Called
MAIN                  10.108    ( 22.10%)      10.688    ( 31.60%)       1
  SUBRAD               0.065    (  0.14%)       0.069    (  0.20%)      38
    CLOUDR             0.368    (  0.80%)       0.394    (  1.16%)      19
      CLOUDR/p*        0.009    (  0.02%)       0.009    (  0.03%)      19
    PRERAD             0.145    (  0.32%)       0.159    (  0.47%)      19
      IRRAD            0.634    (  1.39%)       0.666    (  1.97%)    1824
        IRCLD          0.679    (  1.48%)       0.759    (  2.24%)     507
          IRCLD/EMISC  3.812    (  8.33%)       3.866    ( 11.43%)   20280
          IRCLD/EMISC2 1.948    (  4.26%)       1.973    (  5.83%)   12168
          IRCLD/FLUXC  0.879    (  1.92%)       0.913    (  2.70%)   20280
          IRCLD/FLUXC2 0.433    (  0.95%)       0.454    (  1.34%)   12168
        IRCLR          1.754    (  3.83%)       1.995    (  5.90%)    3312
          IRCLR/EMIS  20.071    ( 43.88%)       6.319    ( 18.68%)   88140
          IRCLR/FLUX   4.180    (  9.14%)       4.887    ( 14.45%)   88140
        IRRAD/SOLAR    0.657    (  1.44%)       0.674    (  1.99%)    1824
    ZENITH             0.002    (  0.01%)       0.003    (  0.01%)      19
* TOTAL               47.364                   33.826                248758
```

Interesting points we can see in the timings are the following.

1. The radiation calculation takes almost 80% of the total time for a full radiation time step, even though only 25% of the points include the radiation calculation.
2. The IRCLR/EMIS (DO 4020 loon) accounts for more than half of the radiation calculation. The emissivity and flux calculations together account for 2/3 of the entire time.
3. Horner's rule speeds up the DO 4020 loop by a factor of more than 3, and speeds up the entire radiation calculation by a factor of 1.6.
4. The entire solar calculation is only 1.5% of the total time.
5. The calculation of surface pressure from the spectral coefficients is trivial.

Space and time interpolation

The radiation calculation is the most time consuming part of the model. Consequently, a full radiation calculation is only performed every three hours, every other latitude and every other longitude. The heating rates so calculated are held constant for the next three hours, the next longitude and the next latitude. The idea of calculating the full radiation only for some time steps and some grid points is generally accepted, however a better space and time interpolation scheme must be used to justify this approach and even then the justification is weak, when one considers the effect of clouds.

We conjecture, that with a better interpolation scheme, radiative heating rates as accurate as those now obtained could be obtained with full radiation calculations on a much coarser time and space grid. For example, the steps in the heating rate every other grid point in the current scheme will, when spectrally filtered, introduce some very high spatial frequency variability in the radiative heating rates. We do not plan to implement this potential speed-up, because doing so would violate our general strategy to avoid impacts. In this case it would be hard to verify that the heating rates are indeed more accurate. However, the following discussion is offered as a possible future speed up.

We recognize that the principal requirement for high resolution is the accurate depiction of the advective processes. The useful resolution of any model is much less than its actual resolution. The dynamics of the primitive equations, as well as the explicit model filters, tend to remove high temporal and horizontal frequencies. Since the radiative heating enters the model as a linear forcing term a reasonable approximation would be to remove high frequencies from the radiative heating directly. Furthermore, for NWP radiation is secondary, at least for the bulk of the atmosphere, where heating rates are small or equivalently slowly acting. Therefore as long as the average heating rates are correct, our approximations should be acceptable.

In order of importance radiative fluxes depend on clouds, especially cloud amount and height, solar zenith angle, temperature, and absorber amount (for current purposes specific humidity). Clouds produce great variations in both the thermal and solar fluxes. Therefore we might calculate fluxes separately for the different cloud formations and perform areal averages at each grid point at each time based on the actual cloud amounts. This approach is especially attractive if we are willing to limit the number of cloud formations to two - clear and cloudy - as in the current code.

Variations in the source functions can be large over the course of several hours - in the solar as the sun rises or sets and in the thermal as the surface and lower atmospheric layers warm or cool. Variations with location of both temperature and solar zenith angle are slow and could be well represented on a coarse grid. The temporal variations are to first order accounted for in the ECMWF model by defining an effective emissivity ϵ_e and transmissivity τ_e for each model level such that $F_T = \epsilon_e \pi B(t)$ and $F_S = \tau_e S_0$ where F_T and F_S are the net thermal and solar fluxes, $B(T)$ is the Planck function for the temperature T and S_0 is the solar flux incident at the top of the atmosphere. The values of ϵ_e and τ_e are kept constant between full radiation time steps and the net fluxes are recomputed at every time step using the above expressions with the current temperature and solar zenith angle (ECMWF, 1988). This approach takes into account only the zeroth order effect of the variation of temperature and zenith angle. That is, the solar heating is proportional to the input solar radiation and the thermal cooling at a level is proportional to T^4 at that level. The ECMWF approach does not account for changes in cloudiness which take place between full radiation calculations.

Variations in fluxes due to the variations in gaseous absorber amounts should be small. Of course humidity varies rapidly in space and time, but its effect on fluxes is secondary compared to cloud effects and the source terms. Because of the spatial variation of humidity and surface albedo, it is desirable that the horizontal interpolation include a spatial filter which does not alias the short wavelengths.

In the ECMWF model, the horizontal interpolation is zonal only. For each latitude variables needed by the radiation calculation are transformed to a coarser subgrid via Fourier filtering. On this coarser grid, fluxes are then evaluated via the complete radiation algorithm. Then, effective transmissivities and emissivities are computed and Fourier transformed back to the full resolution of the model (ECMWF, 1988). The interpolation could be done globally, using spherical harmonic transforms instead of FFTs. This would require storage for the spherical harmonic coefficients, appropriate for the coarse grid, of the radiative parameters. These coefficients would be recalculated at the start of each full radiation time step. Then during each time step, these coefficients would be evaluated on the Gaussian grid used in the nonlinear dynamics calculation and used to calculate heating rates as described above. We might calculate the values of ϵ_e and τ_e separately for different cloud formations on the coarse grid and then at each time and each location calculate the fluxes for each cloud formation and average by area as outlined previously.

Thermal calculation in cloudy atmospheres

For IR radiative transfer, for a particular band, at a particular level, the upward flux is given by the source term at the boundary beneath the level times the transmission from the boundary plus a sum over the intervening layers of the source term evaluated in the layers times the difference in transmissions from the bounding levels. Assuming no scattering, transmission is equal to 1 - emission. This gives us equation B.1 of (Ou and Liou, 1988),

$$F_k = \overline{B}_s T_{sk} + \sum_{j=s+1}^{k} B_j \left(T_{jk} - T_{j-1,k} \right) \tag{7}$$

where F_k is the upward flux at level k, $\overline{B}_s$ is the source function evaluated at level s, B_j is the source function in layer j and T_{ik} is the transmission from level i to level k. B is essentially the Planck function, but is taken to be simply $\sigma\theta^4$, with the band constants absorbed into the transmissivities or emissivities. Thus B does not depend on the band. Furthermore, (almost) the only effect of clouds is through B, when clouds are black bodies.

Of course clouds are not black bodies, but the calculation proceeds as if they were black bodies and then certain corrections are added to the fluxes. For the present implementation there is no correction for black clouds, i.e. low and middle, although Ou and Liou discuss a correction to the intra-cloud net fluxes. The correction for cirrus depends only on the upward flux at the cirrus base and the downward flux at the cirrus top. (These are at fixed levels.) These particular fluxes for the cloudy part of the scene will need to be saved for the correction.

Returning now to the pure black cloud case, we adjust B in order to average over the cloud formation (Hoffman, 1981; Dopplick, 1972; Rodgers and Walshaw, 1966). The contribution from any single cloud formation can be extended to a sum over all layers by defining an adjusted source function:

$$F_k = \overline{B}_s T_{sk} + \sum_{j=s+1}^{k} B_j \left(T_{jk} - T_{j-1,k}\right) \tag{8}$$

$$= \overline{B}_s \left[T_{0k} + \sum_{j=1}^{s} \left(T_{jk} - T_{j-1,k}\right) \right] + \sum_{j=s+1}^{k} B_j \left(T_{jk} - T_{j-1,k}\right) \tag{9}$$

$$= \tilde{\overline{B}}_0 T_{0k} + \sum_{j=1}^{k} \tilde{B}_j \left(T_{jk} - T_{j-1,k}\right) \tag{10}$$

where $\tilde{\overline{B}}_0 = \overline{B}_s$ and $\tilde{B}_j = B_j$ for $j>s$ and $\tilde{B}_j = \overline{B}_s$ otherwise. This sum is the same for each cloud formation except that $\tilde{B}$ and $\tilde{\overline{B}}$ change. Further, since F_k is linear in $\tilde{B}$ and $\tilde{\overline{B}}$, to average over the cloud formations we may simple average over the source functions. An efficient algorithm for accomplishing this is provided in Appendix B. Note that except for calculating the averaged adjusted source function (denoted c in the appendix), the calculation is no different from a single clear column calculation.

There are several advantages to this approach: It eliminates redundant calculations of transmission. Since the transmissions include the band constants the average over cloud formations is all accomplished by averaging the σT^4, which are independent of frequency. The cloud effect on the algorithm is isolated.

Solar calculation in cloudy atmospheres

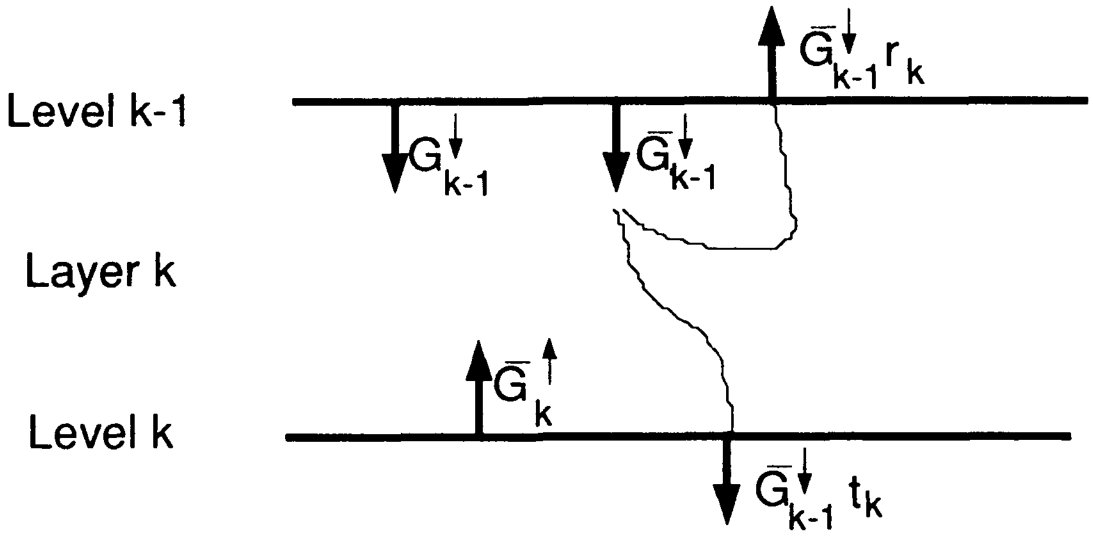
We wish to use the same formalism for the cloudy case, whether there are 1, 2 or 3 clouds present. For this purpose we group the model layers into 5 superlayers, namely the three cloud decks, and the clear atmosphere at the top and bottom of the atmosphere. We number these superlayers from 0 through 4 starting at the top of the atmosphere. We will use the idea of generation functions (also known as the adding method) to determine the various fluxes at all the model levels.

From now on we consider only the problem of determining the generation functions and we refer to these superlayers as layers. The approach taken could be applied directly to the layers or the superlayers. However the superlayer approach maintains continuity with the present code. Further, once we solve for the fluxes at the boundaries of the superlayers, it is very simple to determine the remaining fluxes. The generation functions are the fraction of the total flux entering the system which 'leaves' a particular level in either the upward or downward direction. We will refer to the generation functions as fluxes.

Regardless of how many clouds are present, for each layer we define layer transmissivities for direct and diffuse radiation and layer reflectivities for direct and diffuse radiation. For clouds the transmission includes forward scattering. If a layer is clear, its reflectivity is zero and its transmissivity is simply 1 - A, where A is the layer gaseous absorption.

For each layer k there is downward direct flux $G^{\downarrow}_{k-1}$, downward diffuse flux $\overline{G}^{\downarrow}_{k-1}$ and upward diffuse flux $\overline{G}^{\uparrow}_{k}$. Bear in mind that the G are total fluxes including multiple reflections.

Figure 5 Fluxes entering layer *k*. The components of the flux leaving the layer due to the downward diffuse flux entering the layer are shown.



Level k-1
$\bar{G}^{\downarrow}_{k-1} r_k$
$G^{\downarrow}_{k-1}$
$\bar{G}^{\downarrow}_{k-1}$
Layer k
$\bar{G}^{\uparrow}_{k}$
Level k
$\bar{G}^{\downarrow}_{k-1} t_k$

These three fluxes are shown in the Fig. 5. Since energy is conserved, we trace all these fluxes entering layer *k* and sum the components leaving to obtain relationships between the fluxes leaving and entering. Each flux entering the layer is either reflected back across the level of origin, transmitted to the next level or absorbed. In Fig. 5 this is shown for the downward diffuse flux. The upward diffuse flux is the same upside down with different indexing. The downward direct flux is analogous to the downward diffuse except that while the reflected direct flux joins the upward diffuse flux, the transmitted direct flux is assumed to be entirely direct. More correctly a fraction of it should join the downward diffuse flux, but the parametrization of transmittance is for the total layer transmittance (Liou and Wittman, 1979). Adding the components yields the fluxes out of layer *k*:

$$G^{\downarrow}_{k} = t_k G^{\downarrow}_{k-1} \tag{11}$$

$$\overline{G}^{\downarrow}_{k} = \qquad\qquad \bar{t}_k \overline{G}^{\downarrow}_{k-1} + \bar{r}_k \overline{G}^{\uparrow}_{k} \tag{12}$$

$$\overline{G}^{\uparrow}_{k-1} = r_k G^{\downarrow}_{k-1} + \bar{r}_k \overline{G}^{\downarrow}_{k-1} + t_k \overline{G}^{\uparrow}_{k}. \tag{13}$$

At the upper boundary, the extraterrestrial flux is known and at the lower boundary, the surface reflectivity is known, so that

$$G^{\downarrow}_{0} = 1 \tag{14}$$

$$\overline{G}^{\downarrow}_{0} = 0 \tag{15}$$

$$\overline{G}^{\uparrow}_{N} = r_s G^{\downarrow}_{N} + \bar{r}_s \overline{G}^{\downarrow}_{N}. \tag{16}$$

The direct components may be solved for directly. The remaining equations form a banded system which can be solved almost as easily as a tridiagonal system. (See Appendix C.)

The solar calculation may be skipped on the nightside. When Greenwich is dark the day points are contiguous, otherwise they are divided by Greenwich into two batches. Some latitudes are all night or all day. Consequently we will make the solar calculation a subroutine called with limits on longitude and make 0, 1 or 2 calls as necessary. The alternative would be to always perform the calculation, with zero input solar flux for night conditions.

Implications for cloud modeling

The modifications we plan to make to the code for vectorization should, besides speeding up the radiation calculation, make it feasible and relatively efficient and straightforward to improve the modeling of clouds. In general, we have tried to isolate the calculations particular to the cloudy case and to treat the effect of clouds more generally, so that the main calculation can proceed in the same manner at each grid point, independently of the number and location of the clouds. Therefore the vectorized code should be useful for investigating the effect of different cloud parameterizations, the number of cloud levels, the number of cloud decks, allowing different cloud amounts in different decks or different layers and making different overlap assumptions.

A simple extention which would allow a great deal of experimentation and flexibility would be to allow more than two cloud formations. Considering that clouds usually appear in no more than three layers, a total of 8 cloud formations would be sufficient for all cases. For example with random overlapping of high (H), middle (M) and low (L) clouds the 8 formations would be clear, H, M, L, HM, HL, ML and HML. In the thermal calculation this change would complicate the calculation of the adjusted source function, and require saving the high cloud fluxes for four cloud formations.

In the solar, the entire calculation might be redone for each cloud formation. With the random overlap assumption, only a single calculation of the sort outlined would be required for any number of layers having different cloud amounts. An alternative used in the ECMWF model (ECMWF, 1988) is to formulate the adding method more generally. In their approach, it is assumed that radiation leaving the clear (or cloudy) part of a layer is independent of its previous history. For example, consider a cloudy part of a layer: In general, radiation enters this part from the clear and cloudy parts of the adjacent layers, but the radiation leaving this part is considered as a single item. The equations relating the fluxes entering and leaving a layer are similar to those discussed above, but now there are four fluxes at each interface and it is uncertain whether a solution as simple as that described in Appendix C could be obtained.

4.3 Boundary layer

Current status

The main subroutines of the OSU planetary boundary layer (PBL) and ground parameterizations are described here. The interface routine for the PBL and ground computations is subroutine BLFLUX, which is called once per time step at each latitude pair (northern and southern hemisphere). The time stepping is a forward implicit time step from *t-1* to *t+1*. In the ground, the temperatures and moistures are time filtered.

BLFLUX

At the first call, some constants are set up through a call to BLINIT. The maximum height of the PBL is the first level above 500 mb. At the start of BLFLUX, the surface grid point values (at time t-1) are computed by calling GETGP1. A double outer loop, over hemispheres and over longitudes, surrounds the computation. Within this loop, data are extracted from the model 3d variables (time t-1) to create a 1d column with the vertical index increasing upwards (inverse of the 3d variables). The height of the levels is computed in SIGTOZ, by solving the hydrostatic equation. Routine SFLX is called to compute the ground variable and the surface fluxes. Routine PBLL is called to compute the PBL tendencies. Diagnostic printing is done. Finally the 1d PBL tendencies are added to the 3d tendencies of the other processes.

SFLX

Subroutine SFLX is the outer routine for all the surface and ground parametrization routines. First the drag coefficient is computed. Then routine HFLX is called to compute the ground heat flux at the surface. If there is snow, that flux is computed here in SFLX. The potential evaporation is then computed, using the Penman equation. Routine SMFLX is then called to compute surface moisture, runoff, ground and canopy moisture processes. SMFLX also computes the evaporation, but it is not used in the PBL routine (it is recomputed in PBLL). It is only used here for the surface energy balance. Finally, KTSOIL is called to compute the soil diffusivity and SHFLX is called to compute the ground heat flux and soil temperatures.

PBLL

Subroutine PBLL first computes the height of the PBL. An interpolation is used within the lowest layer in which the Richardson number becomes greater than a critical value. Then the computation is divided between stable and unstable cases. In stable cases, clouds are computed, using a quadratic function of relative humidity above 80%, with maximum overlap, then PBL temperature and humidity are scaled and a coefficient for the diffusion coefficient calculation is computed. In unstable cases a new PBL height is computed, using a countergradient flux effect, then a different cloud computation is performed (discussed below). There is a provision for a modification of the diffusion coefficient in clouds, but in the current version this computation, although performed, has no effect. In a new loop over levels, the diffusion coefficient is computed, differently for stable and unstable cases. Finally, the diffusion equation is solved (tri-diagonal matrix inversion), and the tendencies computed.

Timings

As was seen in table 2, the PBL computations (including transforms) take up 4% of the time step with radiation (27% for a time step without radiation). Fig. 6 shows a further breakdown of that figure (excluding the transforms); percentages shown there are for a time step with radiation.

It can be seen that the boundary layer (PBLL) and soil (SFLX) both contribute significantly to the total time spent. In the boundary layer, the lifting condensation level (LCL) computation represents roughly 12% of the CPU time. It should be noted that the LCL is only used to define the PBL cloud layer. Since the option to have the diffusion coefficients affected by the clouds is not used, computation of the LCL could be avoided entirely. In the soil computations, the computation of the diffusivities (KTSOIL and DFKT) are the single most expensive calculations.

Figure 6 Timings of the current PBL routines. Times are shown for the first time step (with radiation). The routines are hierarchically ordered in a call tree (and in alphabetical order at the same level of the call tree). If a routine appears more than once in the tree, only the first appearance is shown. The transforms (GETGP1 and routines called by it) have been omitted from this list. The time spent in the saturation vapor pressure routines (SVP and DQSDT) is included in the time of the calling routines.

```
Routine                 Time executing   Called

BLFLUX                 0.228   (  0.50%)     38
   BLINIT                  >   (  0.00%)      1
      HFAK                 >   (  0.00%)      1
   PBLL                0.865   (  1.92%)   7296
      LCL              0.142   (  0.31%)   3440
         TDEW          0.055   (  0.12%)   3462
   PRINT               0.007   (  0.02%)      6
      SP                   >   (  0.00%)     48
      TOPQ                 >   (  0.00%)      7
   SFLX                0.196   (  0.44%)   7296
      HFLX             0.012   (  0.03%)   6665
         KTSOIL        0.109   (  0.24%)  20998
      SHFLX            0.035   (  0.08%)   3834
         HRT           0.042   (  0.09%)   3834
         HSTEP         0.018   (  0.04%)   3834
            ROSR12     0.036   (  0.08%)   7668
      SMFLX            0.064   (  0.14%)   3834
         SEC           0.006   (  0.01%)   2444
         SEDIR         0.011   (  0.02%)   2444
            DFKT       0.107   (  0.24%)  10114
         SET           0.016   (  0.03%)   2444
         SRC               >   (  0.00%)   3834
         SRT           0.037   (  0.08%)   3834
         SSTEP         0.026   (  0.06%)   3834
         THSAT           >>>   (  0.00%)      1
   SIGTOZ              0.043   (  0.10%)   7296
```

Strategy

Our general strategy will be similar to what we described for the other parts of the code: eliminate unnecessary computation (like multiplying, then dividing by Δt) ; eliminate all redundant IF statements Replace all GOTO structures by IF/THEN or DO structures; put stable and unstable computations in the same loops, with IF statements to separate the two cases; make all vertical loops work on fixed range (either KP, the total number of layers, or MZ, the maximum number of layers allowed for the PBL), using flags to define layers or regions (PBL top, cloud layer); use the 3d variables themselves rather then generating special PBL variables; finally, make inner loops over longitude points. We list additional specific suggestions below.

The SFLX routine contains some complicated logical structures to allow options such as suppressing the snow computation, or suppressing the surface heat flux computation entirely. These structures should be eliminated.

In PBLL, the two loops that compute the PBL height can be replaced by a single one as follows:

```
      if (heatv.gt.0.) then
         thl=tsv
      else
         thl=tbl(2)
      endif
      flagh=1
      rih=-g*heatv*pr*zz2/(t2*ustar2*vv)
      if(rih.gt.ricr)flagh=0
      jm=0
      rim=0
      rip=0
      do 200 j=3,kp
         if (heatv.gt.0.) then
            cgth=heat*fak/(ustar*(1-binm*zz(j)/xl)**onet)
          cgth=amin1(cgth,6.5E-4*zz(j))
          tlv=t2v+cgth
       else
          tlv=t2v
       endif
       flagl=flagh
       ril=rih
       vvj=ubl(j)*ubl(j)+vbl(j)*vbl(j)
       tjv=tbl(j)*(1.+0.61*wbl(j))
       rih=g*(tjv-t2v)*zz(j)/(thl*vvj)
       if(rih.gt.ricr)flagh=0
       rim=rim+(flagl-flagh)*ril
       rip=rip+(flagl-flagh)*rih
       jm=jm+(flagl-flagh)*(j-1)
  200 continue
      if(rim.eq.0.and.rip.eq.0)jm=2
      if(rim.eq.0.and.rip.eq.0)rim=ricr
      hpbl=zz(jm)+(ricr-rim)*(zz(jm+1)-zz(jm))/(rip-rim)
      jpbl=min(mz,jm)
C   If we want to limit hpbl to z(mz), then:
      hpbl=min(hpbl,zz(mz))
```

A similar type of structure can be used 'o compute the diffusion coefficients. That computation can be made for all levels if K is set to 0 or some small value for levels above the PBL height hpbl.

Diagnostic printing should be done in a special I/O routine. It should not be resolution dependent as it is now.

The call to GETGP1 to compute grid point values of the surface variables from their spectral coefficients could be done outside of the PBL routines. That would make the code cleaner.

The computation of cloudiness in unstable cases is very strange. It starts like a quadratic relationship with the mean relative humidity in the cloud layer (with coefficients defined with the remarkable accuracy of 10^{-8}!) but then a correction is made so that there is a jump of 1.3% cloudiness at the critical relative humidity of 57%. Since PBL clouds are not used, this computation could be avoided altogether.

Allowance is made, in the code, for the possibility of modifying the diffusion coefficients in the presence of clouds. Setting IFCLD to 0 (in a DATA statement in BLOCK DATA A4) makes quite a few computations unnecessary. They should be eliminated from the code.

Variables that are not used such as UNM, VNM, RNM, QNM in common block /FIELDS/ should be eliminated.

Both SHFLX and SMFLX start with a check on the number of ground layers. This test should be done only once, if at all, in the set-up routine.

The two routines that deal with the canopy (SEC and SRC) are both one line routines and can be joined into a single routine. Routine HFLX, which computes the ground heat flux at the surface, needs to be computed only once per time step (and not twice as it is now) if its results are stored for the next time step.

Routine SVP, to compute saturation specific humidity, will fail if $e_s > p/.378$ (watch out in stratosphere). This routine is called repeatedly for the same temperatures and pressures (*t-1* time level). It should be done only once, before the physics routines are called, and the results stored.

The flow diagram of the revised PBL scheme is shown in Fig. 7

4.4 Adjustment physics

The adjustment physics is invoked by a call to CONPCP in each time step. Within CONPCP, preliminary spectral values of temperature, moisture, and ln (surface pressure) are transformed to gridspace in a loop over latitude analogous to the latitude loop for the calculation of the tendencies. For each latitude, the three adjustment routines for moist convection (MODKUO), large scale precipitation (LRGSCL), and dry convection (FIXTET) compute adjusted gridpoint values of temperature and moisture. The values are then transformed back to spectral space. Upon completion of the latitude loop, the time filter is applied to temperature and moisture.

For the existing GSM, the CONPCP subroutine requires approximately 20% of the total CPU time for a time step without radiation; this number drops to 3% for a time step with radiation. The time-filter, which is more properly considered a part of the adiabatic calculations, only requires 1% of the time spent in CONPCP. Of the remainder, 50% is spent on the transforms, 43% in MODKUO, 5% in FIXTET, and 2% in LRGSCL. Thus, aside from the transforms, which are discussed elsewhere, by far the major part of the CPU time is spent in moist convection, followed by dry convection and large scale precipitation. Our emphasis for optimization will thus be the moist convection.

We propose to retain the CONPCP routine as the interface between the adiabatic calculations and the adjustment physics. It will be changed to allow implementation of the general truncation

Figure 7 Flow diagram of the revised PBL scheme

```
(A) Initialization

    BLINIT -  (called once, to initialize constants)
           - HFAK

(B) Preliminaries
    GETGP1 - (get surface variables grid point values)
    (analogous to, or incorporated into, SPTOGP)
    CSVP   - (compute saturation moisture at t-1)

(C) Plug-compatible module
    BLFLUX - (control routine)
           - SIGTOZ - (height of levels)
    - SFLX   - (controls soil computation)
      - DQSDT  - (derivative of qs wrt temperature)
               - SVP
      - SMFLX  - (soil, canopy moisture computation)
        - DFKT   - (soil moisture diffusivity)
        - SEDIR  - (direct evaporation (bare soil))
        - SET    - (plant transpiration)
        - SEC    - (canopy evaporation)
        - SRT    - (rhs of soil moisture equation)
        - SSTEP  - (do time step for soil moisture)
          - ROSR12 - (solve 3-diagonal matrix)
      - KTSOIL - (soil heat diffusivity)
      - SHFLX  - (soil temperature computation)
        - HRT    - (compute matrix elements)
        - HSTEP  - (do time step)
          - ROSR12 - (solve 3-diagonal matrix)
        - HFLX   - (soil heat flux at surface)
    - PBLL   - (PBL computation)
                    - LCL    - (lifting condensation level)
                      - TDEW   - (computes dewpoint temperature)
    - DIAG   - (diagnostics printed as needed)
```

version of the transform routines, and the time filter of the adjusted values will be moved to the calling routine. The current practice of using edit symbols for array dimensions will be replaced by PARAMETER statements. The loop over latitude will be multiprocessed in the same way as in the calculation of the tendencies (except that it is not necessary to treat two latitude pairs as one task). The flowchart of the revised CONPCP routine is shown in Fig. 8.

Additional minor changes will be necessary to the calls of the three adjustment subroutines, to allow closer adherence of those routines to the plug-compatibility rules discussed earlier in this report. To preserve data integrity all three routines will be changed to eliminate reads and writes (R/W) to global accumulators - instead, these variables will be made latitude dependent (and thus write-only (W) inside the subroutines), and summed within CONPCP. Physical constants will be either passed in as arguments from CONPCP, or initialized in a separate initialization

Figure 8 Proposed design of the adjustment physics

```
CONPCP -
        ----------------- Begin - Latitude - Loop ---------------------

        (A) - spectral to gridpoint transformation:
              (analogous to the hydrodynamics)

        (B) - adjustment physics:
        - MODKUO (moist convection - Kuo)
        - LRGSCL (large scale precipitation)
        - FIXTET (dry adiabatic adjustment)

        (C) - gridpoint to spectral transformation:
              (analogous to the hydrodynamics)

        ------------    - End - Latitude - Loop ---------------------
```

routine and passed via global commons. In simple enough cases, they may also be defined via PARAMETER statements. No SAVE statements will be used in the code. Local work arrays will be dimensioned by parameter statements - no edit symbols will be used in the plug-compatible modules. We now turn to a discussion of the individual routines.

MODKUO

The moist convection routine has an outer loop on longitude, with all calculations done for one vertical column at a time. For each vertical column, the moisture convergence is computed; no convection calculation is performed if the moisture convergence is too small.

The in-cloud profile of temperature and moisture are determined in a two-step process. First, the lifting condensation level (LCL) and the equivalent potential temperature at the LCL are computed in subroutine LCLAFGL. The lowest model layer with a central pressure less than the LCL pressure is considered the cloud bottom. Next, for layers starting from cloud bottom, the temperature and moisture corresponding to the previously computed equivalent potential temperature are computed in subroutine SUDOAD. Cloud top is defined as the highest unstable layer, except that one stable layer (in-cloud temperature less than the ambient temperature) is permitted to interrupt a sequence of unstable layers.

Following determination of cloud top, the preliminary values of temperature and moisture are adjusted by mixing with the corresponding in-cloud values for layers between cloud top and bottom, provided the cloud is deep enough. Provisions are made to consider entrainment of environmental air into the cloud, compute the partitioning of the converged moisture for heating and moistening, and consider the evaporation of falling precipitation.

A breakdown of the CPU time spent in MODKUO reveals that 81% is spent in the subroutine SUDOAD (and routines called by it), and another 9% in LCLAFGL. Thus fully 90% is spent for the computation of the in-cloud profile of temperature and moisture. On a global basis, approximately 14% of all gridpoints have enough moisture convergence so that LCLAFGL is called; of those, 59% have no conditional instability (cloud top = cloud bottom), and only 41% (6% of all points) have an adjustment calculation. After spinup of the convection (typically several hours into a forecast), the percentage of points with convection will be higher. As is to be expected, more points are unstable (and clouds are deeper) in the tropical latitudes, and the high latitudes (roughly 20% of all latitudes) have no convection at all.

Vectorization of the existing code is difficult because both LCLAFGL and SUDOAD use an iterative procedure to determine the points on the moist adiabat. Up to 10 (20 in the case of SUDOAD) iterations are performed, until the procedure converges to a specified accuracy. In each iteration, the saturation vapor pressure is computed by a call to SVP, and the derivative of the saturation mixing ratio with respect to temperature is computed by a call to DQSDT. SVP and DQSDT use a 6th order polynomial developed by Lowe (Lowe, 1977) if the temperature is above -100° C (with different coefficients for temperatures above and below -50° C), and an exponential otherwise.

The code will be reorganized to have inner loops on longitude. Logical flags will be maintained to skip unnecessary computations (this is necessary since moist convection is taking place in only a small fraction of gridpoints). For this reorganization it is necessary to revise the computation of the in-cloud temperature and moisture profiles.

We will replace the present implicit calculations with an explicit ascent calculation. The ascent calculation will proceed in 2 stages: first, the change in temperature due to adiabatic cooling will be calculated; then, if supersaturation is detected, temperature and moisture values will be adjusted toward saturation. The first stage is a straightforward explicit calculation, using:

$$T_k^0 = T_{k+1}^c \left(\frac{p_k}{p_{k+1}} \right)^{\frac{R}{c_p}} \quad ; \quad r_k^0 = r_{k+1}^c \tag{17}$$

where T is temperature, r the water vapor mixing ratio, p atmospheric pressure, R the gas constant and c_p the heat capacity for dry air. The subscript indicates the vertical layer (increasing downward), and the superscripts denote the cloudy value at the previous layer (c) or the preliminary cloudy value at the current layer (0). We note that the factor

$$\left(\frac{p_k}{p_{k+1}} \right)^{\frac{R}{c_p}} = \left(\frac{\sigma_k}{\sigma_{k+1}} \right)^{\frac{R}{c_p}} \tag{18}$$

can be rewritten in terms of the vertical coordinate σ and thus be precalculated for all vertical layers. The adjustment toward saturation will use a wet-bulb adjustment, similar to the formula currently used in LRGSCL:

$$\begin{aligned} r_k^{\tau+1} &= r_k^\tau - \frac{r_k^\tau - r_{sat}(T_k^\tau, p_k)}{1 + \frac{L}{c_p} \frac{dr_{sat}}{dT}(T_k^\tau, p_k)} \\ T_k^{\tau+1} &= T_k^\tau + \frac{L}{c_p} (r_k^{\tau+1} - r_k^\tau) \end{aligned} \tag{19}$$

Here L is the latent heat of vaporization, r_{sat} the saturation value of r, and the superscript indicates the iteration number. This adjustment will be applied a fixed number of iterations at all saturated points. The number of iterations will be an adjustable PARAMETER; we plan to use two iterations of the adjustment (the resulting inaccuracies are expected to be small compared to the errors introduced by neglect of other cloud processes, such as water/ice phase changes). We will perform tests with more iterations and evaluate the impact on the solutions.

Cloud top and bottom will be defined as before, and flags will be maintained to indicate whether a particular layer is cloudy, to avoid unnecessary calculations in the adjustment of the environmental temperature and moisture values.

To allow vectorization of the saturation adjustment calculation, the current formula for the saturation vapor pressure must be replaced with a simpler formulation. We tested two alternatives to the Lowe formula (Lowe, 1977): the exponential due to Tetens (as given in (Lowe, 1977)), and a formulation by Derickson and Cotton (Derickson and Cotton, 1977). In the latter, the exact Goff-Gratch formula (as given in (Murray, 1967)) is used for integer values of absolute temperature, and a second-order Taylor series for values in between. The values of saturation vapor pressure, along with its first and second derivatives with respect to temperature, are calculated once for the integer values of absolute temperature, and then used in a simple interpolation formula. Both alternatives were found to vectorize and execute faster than the current routine, with the Derickson and Cotton method being the fastest and most accurate of all three (the Tetens formula is slightly less accurate than the current routine in the temperature range -100° C < t < +50° C). Therefore, we propose to use the Derickson and Cotton method.

FIXTET

The dry convection code was used as a test bed for our vectorization strategy, since it was relatively short, yet contained some of the same complications and impediments to vectorization as the moist convection code. In particular, the original code was organized with an outer loop over longitude, and short inner loops over the vertical index. The loop limits of the inner loop depended on longitude.

We have rewritten and tested this routine for complete vectorization over longitude as the innermost loop. We found that it was necessary to revise the algorithm used in this routine in order to achieve any speedups from vectorization over longitude. Speedups for this (not very CPU-intensive) routine were relatively modest (6% to 27%, depending on the number of longitude points). Both the baseline code and our modifications to it are described in detail in Appendix A; we will implement version 1.6 described there in the GSM.

LRGSCL

This code is already structured in a way to allow efficient vectorization: the innermost loops are over longitude, and they vectorize both on the Cray 2 and the Alliant. Thus no major changes are necessary to this routine.

5 Summary and future enhancements

We have described our multitasking and vectorization design for the GSM. We propose to implement the latitude tasking scheme (Hoffman and Nehrkorn, 1989) for multiprocessing the loop over latitude in the calculation of spectral tendencies and adjusted model variables. The general truncation version of the hydrodynamics code (Kaplan *et al.*, 1985) will be used. Wavenumber calculations will be multiprocessed and vectorized over wavenumber. All gridpoint calculations will be vectorized over longitude. The physics packages will be modified to bring them into close compliance with the "plug compatibility" rules (Kalnay *et al.*, 1989).

During the analysis of the current GSM, and the formulation of the proposed design, a number of possible enhancements to the code and the model formulation were found to be desirable. While we do not foresee being able to implement and test these in the current effort, we list them here as suggestions for future enhancements to the GSM.

The dry adiabatic adjustment should not be performed in those layers at which PBL computations are also performed, since those already simulate the atmospheric response to superadiabatic stratifications (and more accurately so). It would be more consistent to replace the dry adiabatic adjustment with a vertical diffusion computation at all layers - this would be most easily done as part of the PBL computation.

Our changes to the radiation scheme make it possible, at least in principle, to relax some of the current restrictions on the diagnosed clouds, e.g. allowing variable cloud tops and different cloud configurations. Further increases in speed and accuracy could be realized by performing the expensive, but essentially second-order, computations in the radiation scheme less frequently, and using better schemes for space/time interpolation instead. The effects of temperature and cloudiness changes between full radiation computations must be taken into account.

The implementation of the general truncation version of the hydrodynamics make it possible to explore the effects of different truncations. In particular, the effects of changing from the current rhomboidal to a triangular truncation with fewer degrees of freedom (and thus faster execution speed) should be investigated. It might also be desirable to recode the hydrodynamics to make use of the latitude wavenumber tasking approach and to simulate the complex arithmetic in the Legendre sums with real arithmetic.

6 References

S. Brenner, C.-H. Yang, and S. Y. K. Yee. The AFGL spectral model of the moist global atmosphere: Documentation of the baseline version. Technical Report 82-0393, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1982. [NTIS ADA129283].

S. Brenner, C.-H. Yang, and K. Mitchell. The AFGL global spectral model: Expanded resolution baseline version. Technical Report 84-0308, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1984. [NTIS ADA160370].

R. G. Derickson and W. R. Cotton. On the use of finite Taylor's series approximations to certain exponential and power functions employed in cloud models. Atmospheric Science Paper 268, Colorodo State University, 1977.

T. G. Dopplick. Radiative heating of the global atmosphere. *Journal of the Atmospheric Sciences*, 29:1278–1294, 1972.

ECMWF Research Department, ECMWFa. *Research Manual 3, ECMWF Forecast Model Physical Parameterisation*, second edition, January 1988.

R. N. Hoffman and T. Nehrkorn. Multiprocessing a global spectral numerical weather prediction model. In *Fourth Conference on Parallel Processing for Scientific Computing*, Chicago, IL, 1989. Society of Industrial and Applied Mathematics.

R. N. Hoffman. Alterations of the climate of a primitive equation model produced by filtering approximations and subsequent tuning and stochastic forcing. *Journal of the Atmospheric Sciences*, 38:514–530, 1981.

R. N. Hoffman. A simple implementation of the semi-implicit time scheme in a primitive equation spectral model. *Monthly Weather Review*, 113:1829–1831, 1985.

E. Kalnay, M. Kanamitsu, J. Pfaendtner, J. Sela, M. Suarez, J. Stackpole, J. Tuccillo, L. Umscheid, and D. Williamson. Rules for interchange of physical parameterizations. *Bulletin of the American Meteorological Society*, 70:620–622, 1989.

L. D. Kaplan, J.-F. Louis, R. N. Hoffman, R. G. Isaacs, W. J. Gutowski, and T. Nehrkorn. Improving numerical weather prediction by maximizing the use of assimilated satellite data. Technical Report 85-0298, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1985. [NTIS ADA169295].

S. Kinne. *Parameterization of radiative transfer in the Earth's atmosphere with specific applications to clouds*. PhD thesis, University of Utah, Salt Lake City, UT, 1987.

J. M. Levesque and J. W. Williamson. *A Guidebook to Fortran on Supercomputers*. Academic Press, Inc., San Diego, CA, 1989.

K. N. Liou and G. D. Wittman. Parameterization of the radiative properties of clouds. *Journal of the Atmospheric Sciences*, 36:1261–1273, 1979.

K.-N. Liou, S.-C. Ou, S. Kinne, and G. Keonig. Radiation parameterization program for use in general circulation models. Technical Report 84-0217, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1984. [NTIS ADA148015].

Paul R. Lowe. An approximating polynomial for the computation of saturation vapor pressure. *Journal of Applied Meteorology*, 16:100–103, 1977.

L. Mahrt, H.-L. Pan, J. Paumier, and I. Troen. A boundary layer parameterization for a general circulation model. Technical Report 84-0063, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1984. [NTIS ADA144224].

L. Mahrt, H.-L. Pan, P. Ruscher, and C.-T. Chu. Boundary layer parameterization for a global spectral model. Technical Report 87-0246, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1987. [NTIS ADA199440].

F. W. Murray. On the computation of saturation vapor pressure. *Journal of Applied Meteorology*, 6:1410–1411, 1967.

T. Nehrkorn. On the computation of Legendre functions in spectral models. *Monthly Weather Review*, 118:2248–2251, 1990.

S.-C. Ou and K.-N. Liou. Development of radiation and cloud parameterization programs for AFGL global models. Technical Report 88-0018, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1988. [NTIS ADA193369].

C. D. Rodgers and C. D. Walshaw. The computation of infrared cooling rate in planetary atmospheres. *Quarterly Journal of the Royal Meteorological Society*, 92:67–92, 1966.

J. G. Sela. Spectral modeling at the National Meteorological Center. *Monthly Weather Review*, 108:1279–1292, 1980.

S.-T. Soong, Y. Ogura, and W.-S. Kau. A study of cumulus parameterization in a global circulation model. Technical Report 85-0160, Air Force Geophysics Laboratory, Hanscom AFB, MA, 1985. [NTIS ADA170137].

WMO, Geneva. *Report of the Seminar on Progress in Numerical Modelling and the Understanding of Predictability as a Result of the Global Weather Experiment*, Sigtuna, Sweden, October 1984. TD 33. [Publication date 1985].

Appendix A Example of vectorization of adjustment physics routine

A.1 Methodology:

We tested several different versions of the fixtet subroutine in order to evaluate and demonstrate our approaches to vectorization. The tests were performed with synthetic data which had the approximately correct number of unstable points (1/4 of all points was assumed to contain an unstable region consisting of 2, 3, or 4 model layers). We tested the code for 18 layers, with *mp*=96 longitudes (as in the current R30 code) and with *mp*=600 longitudes. Fig. 91 illustrates the distribution of the unstable layers for these two cases.

Figure 9 Distribution of unstable layers among the unstable points. There are 24 (150) unstable points for mp=96 (600). Only the first (topmost) layers are shown; the pattern is repeated for lower layers at larger i.

```
mp=96:

     k\i 1  5  9 13 17 21 25

       1 X  X
       2 X  X  X
       3       X  X  X
       4          X  X  X  X
       5             X  X  X
       6                   X

mp=600:

     k\i 1  5  9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89

       1 X  X  X  X  X  X  X  X  X
       2 X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
       3          X           X     X  X  X  X  X  X  X  X  X  X  X  X  X  X
       4                                  X           X        X  X  X  X  X
       5                                              X           X
```

A.2 Description of codes and test results:

Baseline version (version 1.2):

Before any of the vectorization changes to be discussed, the code was brought into compliance with the minimal requirements for multitasking and plug compatibility. This involved only very minor changes, with no effect on the execution speed. The resulting version (referred to as version 1.2 in the following, shown in Fig. 10) was our baseline version for testing.

Figure 10 Compiler listing (with loop marking) of the baseline version (version 1.2). Initial comment lines and the documentation reference point comments have been removed from the listing. The loop markings are: first letter: S - scalar loop; V - vector loop; W - unwound loop, and second letter: b - bottom loaded; c - conditionally vectorized; r - unrolled.

```
             subroutine fixtet(mp,mpd,kp,kph,tg,qg,teta,ps,del,sl
            &     ,rpirec,rpi,nun,rk)
       c
             dimension  tg(mpd,kp),qg(mpd,kp),teta(mpd,kp),ps(mpd),
            1           del(kp),sl(kp),rpirec(kp),rpi(kp)
             parameter (jl=600,jk=18)
             dimension  work(jl),tet(jk),q(jk)
       c
             if(jl.lt.mp.or.jk.lt.kp) then
                write (*,*) 'Error in fixtet: jl and/or jk too small, abort'
                stop 'Error in fixtet: jl and/or jk too small, abort'
             endif
             kpp1=kp+1
             kph1=kpp1-kph
             sl100k=(sl(kp)/100.)**rk
       c  ...Initialize counter nun for current latitude:
             nun=0
       c***  ps is assumed true surf. press. in cb
       c
V-----<      do 1 i=1,mp
V            work(i) = sl100k*ps(i)**rk
V----->    1 continue
       c     compute bottom pot. temp. in teta(i,kp)
Vr----<      do 2 i=1,mp
Vr           teta(i,kp) = tg(i,kp)/work(i)
Vr---->    2 continue
       c     compute rest of pot temps from bottom one up
S-----<      do 4  kc=2,kp
S            k=kpp1-kc
S            ka=k+1
S Vr--<      do 3  i=1,mp
S Vr         teta(i,k) = tg(i,k)*teta(i,ka)*rpirec(ka)/tg(i,ka)
S Vr-->    3 continue
S----->    4 continue
       c***  now have pot. temp. in teta, adjust.
       c***  qg is mixing ratio
S-----<      do 66666 i=1,mp
S Vr--<      do 5 k=1,kp
S Vr         q(k)=qg(i,k)
S Vr         tet(k) = teta(i,k)
S Vr-->    5 continue
S            itest=0
S      c     tet(k) = pot. temp. at lon.
S S---< 55555 continue
S S --<      do 13  kc=2,kp
S S          k =kpp1-kc
S S          ka=k+1
S S          kk = k
```

```
S S          if (tet(k)-tet(ka)) 14, 13, 13
S S -->   13  continue
S S          go to    26
S S    c
S S       14 kbot = kk + 1
S S          ktop = kk
S S --<   15 sumtet = 0.
S S          sumdel = 0.
S S          sumcol = 0.
S S          sumq = 0.
S S          itest=1
S S    c
S S --<      do 16  k=ktop,kbot
S S          sumdel = sumdel + del(k)
S S -->   16 sumtet = sumtet + del(k)*tet(k)
S S          tet(kbot) = sumtet/sumdel
S S          if (ktop.lt.kphl)  go to 19
S S --<      do 17  k=ktop,kbot
S S -->   17 sumq = sumq + del(k)*q(k)
S S          q(kbot) = sumq/sumdel
S S --<      do 18  k=ktop,kk
S S -->   18 q(k) = q(kbot)
S S          go to    22
S S       19 if (kbot.le.kphl)  go to 22
S S --<      do 20  k=kphl,kbot
S S          sumcol = sumcol + del(k)
S S          sumq = sumq + del(k)*q(k)
S S -->   20 continue
S S          q(kbot) = sumq/sumcol
S S --<      do 21  k=kphl,kk
S S          q(k) = q(kbot)
S S -->   21 continue
S S --<   22 do 23  k=ktop,kk
S S -->   23 tet(k) = tet(kbot)
S S          if (ktop-1) 26, 26, 24
S S--->   24 if (tet(ktop-1)-tet(ktop)) 25, 55555, 55555
S         25 ktop = ktop - 1
S   -->      go to    15
S      c
S         26 continue
S            if(itest)27,66666,27
S      c
S         27 nun=nun+1
S      c     return adjusted potential temps to teta
S Vr--<       do 7 k=1,kp
S Vr-->7      teta(i,k) = tet(k)
S Vr--<       do 8 k=kphl,kp
S Vr-->8      qg(i,k)=q(k)
S      c
S----->66666   continue
       c     convert pot. temps. to temp. in tg(i,k)
       c     start by computing bottom temp. and build up.
Vr----<      do 9 i=1,mp
Vr           tg(i,kp) = teta(i,kp)*work(i)
```

```
Vr---->     9 continue
S-----<       do 11 kc=2,kp
S             k = kppl - kc
S             ka= k+1
S Vr--<       do 10 i=1,mp
S Vr          tg(i,k) = tg(i,ka)*teta(i,k)*rpi(k)/teta(i,ka)
S Vr-->    10 continue
S----->    11 continue
              return
              end
```

The structure of the baseline code is as follows. First, the input values of temperature (*T*) are converted to *θ*. Then, within an outer loop on longitude (*i*), there are numerous inner loops on the vertical layer index (*k*). Initially, local one-dimensional work arrays for theta and *q* are filled with values from the two-dimensional arrays. The adjustment calculations are then performed within a double-loop over *k*: in the outer loop, the lowest two adjacent unstable layers in the column are detected and mixed; the mixing is repeated, if necessary, with layers overlaying the mixed layer until stability is reached. The outer loop is then repeated until there are no more unstable regions. Finally, the one-dimensional arrays are copied to the two-dimensional arrays if any adjustments were performed. After the completion of the longitude loop, all *θ* values are converted to *T*.

All branching within the code is accomplished by arithmetic if-statements.

An obvious problem with this code are the many short loops over *k*, which will not efficiently vectorize. In addition, many computations are needlessly repeated: in the case of *N*-layer unstable regions, mixing computations are performed *N-1* times.

Vectorization changes

As a first step, the arithmetic if statements all throughout the longitude loop were converted to equivalent if-statements or if-then-else structures. This made the code readable (and understandable) enough to allow further optimization. At the same time, the one-dimensional work arrays were eliminated in anticipation of vectorization over longitude. From this intermediate step, three different vectorization approaches were tried.

Direct adaptation of existing algorithm (version 1.4)

Version 1.4 (Fig. 11) is the most straightforward implementation of our vectorization strategy for this algorithm: the innermost loop is longitude for almost all loops over *k* (an exception to this rule is the 200-loop, which involves transfers outside the range of the loop over *k*). Since in the original formulation the loops over *k* only extend over parts of the column (which depend on *i*), an if-check needed to be placed within each loop on *i*. As in the original formulation, all relevant loops over *k* are repeated until there are no more unstable layers at any longitude; flags are set appropriately to only do computations at the appropriate *i*.

Figure 11 As Fig. 10, but for version 1.4. Only the actual adjustment calculation is shown.

```
Vr----<      do 10 i=1,mp
Vr              ktop(i)=kp-1
Vr              itest(i)=0
Vr---->   10 continue
       c
       c     teta(i,k) = pot. temp. at lon.
S-----<      do 500 iter=1,kp-1
S               iany=0
S Sb--<         do 200 i=1,mp
S Sb               if(ktop(i).gt.1) then
S Sb   c           ...Find lowest remaining adjacent unstable layers:
S SbV-<               do 113  k=ktop(i),1,-1
S SbV                     kk = k
S SbV                     if (teta(i,k).lt.teta(i,k+1)) goto 114
S SbV->  113          continue
S Sb                  ktop(i)=0
S Sb                  kbot(i)=0
S Sb                  go to 115
S Sb   c
S Sb     114          kbot(i) = kk + 1
S Sb                  ktop(i) = kk
S Sb                  if(itest(i).eq.0) nun=nun+1
S Sb                  itest(i)=1
S Sb                  iany=iany+1
S Sb     115          continue
S Sb               endif
S Sb-->  200    continue
S               if(iany.eq.0) goto 600
S S---<  201    continue
S S    c     ...Perform adjustment for given tops and bottoms of unstable
S S    c     ...layers
S S V-<         do 210 i=1,mp
S S V              sumtet(i) = 0.
S S V              sumdel(i) = 0.
S S V              sumcol(i) = 0.
S S V              sumq(i) = 0.
S S V->  210    continue
S S    c
S S S-<         do 216 k=1,kp
S S S V-<          do 215 i=1,mp
S S S V               if(k.ge.ktop(i).and.k.le.kbot(i)) then
S S S V                  sumdel(i) = sumdel(i) + del(k)
S S S V                  sumtet(i) = sumtet(i) + del(k)*teta(i,k)
S S S V               endif
S S S V> 215       continue
S S S->  216    continue
S S S-<         do 218 k=1,kp
S S S V-<          do 217 i=1,mp
S S S V               if(k.ge.ktop(i).and.k.le.kbot(i)) then
S S S V                  teta(i,k) = sumtet(i)/sumdel(i)
```

```
S S S V               endif
S S S V->217       continue
S S S->  218    continue
S S S-<         do 220 k=1,kp
S S S V-<          do 219 i-1,mp
S S S V               if(k.ge.max(ktop(i),kphl).and.k.le.kbot(i)) then
S S S V                  sumcol(i) = sumcol(i) + del(k)
S S S V                  sumq(i) = sumq(i) + del(k)*qg(i,k)
S S S V               endif
S S S V->219       continue
S S S->  220    continue
S S S-<         do 230 k=1,kp
S S S V-<          do 229 i=1,mp
S S S V               if(k.ge.max(ktop(i),kphl).and.k.le.kbot(i)) then
S S S V                  qg(i,k) = sumq(i)/sumcol(i)
S S S V               endif
S S S V->229       continue
S S S->  230    continue
S S    c     ...Check for unstable layers adjacent to layers just adjusted,
S S    c     ...repeat adjustment for all adjacent layers if necessary
S S             iany=0
S S V-<         do 300 i=1,mp
S S V              if(ktop(i).gt.1) then
S S V                 if(teta(i,ktop(i)-1).lt.teta(i,ktop(i))) then
S S V                    ktop(i)=ktop(i)-1
S S V                    iany=iany+1
S S V                 endif
S S V              endif
S S V->  300    continue
S S--->         if(iany.gt.0) goto 201
S----->  500 continue
```

This code lead to vectorized inner loops over longitude. However, because not all points with unstable layers at a given *k* are computed at the same time, only very few points within the inner loops over *i* are active, and there is excessive loop overhead. From Fig. 9 it is clear that as few as 1 or 2 for *mp*=96 (and still only 7 or 8 for *mp*=600) points would be active within any given loop over *i*. Because of these problems, this version was significantly slower than the baseline version (see tables 3 and 4).

Single sweep version (version 1.5)

To eliminate the needless repetition of calculations in the baseline algorithm, the double-loop over *k* was replaced with a single loop over *k* (Fig. 12). For each vertical layer, 3 cases are considered:

1. The layer is unstable. In this case the mixed values are computed and stored in the upper layer of the mixed region, and housekeeping parameters are updated accordingly.
2. The layer is stable and adjustment were performed in previous (lower) layers. In this case the adjusted values are stored in all layers of the mixed region, and housekeeping parameters are reinitialized.

Table 3 Fixtet timings for mp=96 and mp=600 for versions 1.2, 1.4, 1.5, and 1.6. Times are CPU times for one latitude, in seconds; shown are the means and standard deviations for 10 runs. Speedup is the speedup compared to version 1.2, and the last column is the ratio of the mp=600 and mp=96 timings.

		CPU time			
Version	mp	mean	stand. dev.	Speedup	Slow-down for mp=600
1.2	96	.01153	.00005	1.0	-
	600	.01887	.00010	1.0	1.637
1.4	96	.01311	.00005	0.879	-
	600	.02662	.00027	0.709	2.031
1.5	96	.01262	.00004	0.914	-
	600	.02624	.00018	0.719	2.079
1.6	96	.01079	.00004	1.069	-
	600	.01396	.00014	1.352	1.294

Table 4 Fixtet timings for mp=96 and mp=600 for versions 1.2, 1.4, 1.5, and 1.6. Shown are CPU time per gridpoint, normalized to the standard case.

Version	mp=96	mp=600
1.2	1.0	0.26
1.4	1.14	0.37
1.5	1.09	0.36
1.6	0.94	0.19

3. The layer is stable and no adjustment were performed in previous (lower) layers. In this case the housekeeping parameters are reinitialized.

Figure 12 As Fig. 10, but for version 1.5. Only the actual adjustment calculation is shown.

```
Vr-----<      do 10 i=1,mp
Vr               nmix(i)=1
Vr               itest(i)=0
Vr               sumdel(i)=del(kp)
Vr----->   10 continue
        c
        c     teta(i,k) = pot. temp. at lon.
S------<      do 300 k=kp-1,2,-1
S                kqtop=max(kph1,k+1)
S S----<         do 200 i=1,mp
S S     c        ...Case 1: Unstable
S S                 if (teta(i,k).lt.teta(i,k+1)) then
S S                    teta(i,k)=(sumdel(i)*teta(i,k+1) + del(k)*teta(i,k))
S S          &                  /(sumdel(i) + del(k))
S S                    if(k.ge.kph1)
S S          &              qg(i,k)=(sumdel(i)*qg(i,k+1) + del(k)*qg(i,k))
S S          &                  /(sumdel(i) + del(k))
S S                    sumdel(i)=sumdel(i)+del(k)
S S                    nmix(i)=nmix(i)+1
S S                    if(itest(i).eq.0) nun=nun+1
S S                    itest(i)=1
S S                 else
S S     c           ...Case 2: Stable, no previous adjustments
S S                    if (nmix(i).eq.1) then
S S                       sumdel(i)=del(k)
S S                    else
S S     c           ...Case 3: Stable, previous adjustments
S S     c           ...Store mixed values in all layers of mixed region:
S S SSb<                  do 110 j=k+2,k+nmix(i)
S S SSb                      teta(i,j)=teta(i,k+1)
S S SSb                      if(j.ge.kph1) qg(i,j)=qg(i,kqtop)
S S SSb>  110             continue
S S                       nmix(i)=1
S S                       sumdel(i)=del(k)
S S                    endif
S S                 endif
S S---->  200    continue
S------>  300 continue
        c
              k=1
              kqtop=max(kph1,k+1)
              kqtop1=max(kph1,k)
S------<      do 400 i=1,mp
S       c     ...Case 1: Unstable
S                if (teta(i,k).lt.teta(i,k+1)) then
S                   teta(i,k)=(sumdel(i)*teta(i,k+1) + del(k)*teta(i,k))
S            &           /(sumdel(i) + del(k))
S                   if(k.ge.kph1)
S            &           qg(i,k)=(sumdel(i)*qg(i,k+1) + del(k)*qg(i,k))
S            &           /(sumdel(i) + del(k))
```

```
S                    sumdel(i)=sumdel(i)+del(k)
S                    nmix(i)=nmix(i)+1
S                    if(itest(i).eq.0) nun=nun+1
S       c        ...Store mixed values in all layers of mixed region:
S SSb--<             do 310 j=k+1,k+nmix(i)-1
S SSb                   teta(i,j)=teta(i,k)
S SSb                   if(j.ge.kphl) qg(i,j)=qg(i,kqtopl)
S SSb-->  310        continue
S                 else
S                    if(nmix(i).gt.1) then
S       c        ...Case 3: Stable, previous adjustments
S       c        ...Store mixed values in all layers of mixed region:
S SSb--<                do 360 j=k+2,k+nmix(i)
S SSb                      teta(i,j)=teta(i,k+1)
S SSb                      if(j.ge.kphl) qg(i,j)=qg(i,kqtop)
S SSb-->  360           continue
S                    endif
S                 endif
S------>  400 continue
```

This code has the advantage of only requiring one set of calculation for each mixed region. However, because of the complicated if-structure and the short inner loops on k, the loop over longitude did not vectorize and the code also executed more slowly than the baseline.

Two-sweep (up and down) version (version 1.6)

This version (Fig. 13) is quite similar to the previous version, except that the computation and the storage of mixed values have been separated into two loops over *k*, allowing vectorization of the inner loops over *i*. This was done at the expense of an additional 2-d work array (LMIXED). The counting of unstable points was also moved to separate loops. As can be seen from tables 3 and 4, this version executes the fastest of all the versions tested, and its execution time increases more slowly with the number of gridpoints.

Figure 13 As Fig. 10, but for version 1.6. Only the actual adjustment calculation is shown. The array lmixed is initialized to .false. prior to the code segment shown.

```
Vr---<      do 10 i=1,mp
Vr             sumdel(i)=del(kp)
Vr--->   10 continue
      c     teta(i,k) = pot. temp. at lon.
S----<      do 300 k=kp-1,1,-1
S V--<         do 200 i=1,mp
S V   c        ...Case 1: Unstable
S V               if (teta(i,k).lt.teta(i,k+1)) then
S V   c     note mixing by marking top layers
S V                  lmixed(i,k) = .true.
S V                  teta(i,k)=(sumdel(i)*teta(i,k+1) + del(k)*teta(i,k))
S V        &              /(sumdel(i) + del(k))
S V                  qg(i,k)=(sumdel(i)*qg(i,k+1) + del(k)*qg(i,k))
S V        &              /(sumdel(i) + del(k))
S V                  sumdel(i)=sumdel(i)+del(k)
S V               else
S V                  sumdel(i)=del(k)
S V               endif
S V-->  200    continue
S---->  300 continue
      c
      c     at this point only the top-most layer of any unstable region
      c     has the correct theta.  same for moisture, except that values
      c     above kphl are meaningless and if a mixed layer straddles the
      c     moist and dry regions then qg(i,kphl) is correct.
      c     now count number of longitude points effected.
V----<      do 410 i=1,mp
V              ltest(i)=.false.
V----> 410  continue
S----<      do 430 k=1,kp-1
S Vr-<         do 420 i=1,mp
S Vr              ltest(i)=ltest(i) .or. lmixed(i,k)
S Vr-> 420     continue
S----> 430  continue
            nun=0
V----<      do 440 i=1,mp
V              if (ltest(i)) nun=nun+1
V----> 440  continue
      c     adjust theta in mixed layers above topmost moist layer
      c     whenever an upper mixed layer is detected propagate down
S----<      do 540 k=1,kphl-1
S V--<         do 530 i=1,mp
S V               if (lmixed(i,k)) teta(i,k+1)=teta(i,k)
S V--> 530     continue
S----> 540  continue
      c     adjust theta and moisture in mixed layers below topmost moist laye
      c     whenever an upper mixed layer is detected propagate down
S----<      do 640 k=kphl,kp-1
S V--<         do 630 i=1,mp
```

```
S V              if (lmixed(i,k)) then
S V                 teta(i,k+1)=teta(i,k)
S V                 qg(i,k+1)=qg(i,k)
S V              endif
S V--> 630     continue
S----> 640  continue
```

A.3 Discussion

Vectorizaiion over longitude resulted in a modest speedup compared to the baseline code (6% to 27%, depending on the number of longitude points), but only after the algorithm used in this routine was revised. In this regard, this routine may represent something of a worst case scenario: in other physics calculations the calculations themselves will be more time-consuming and/or the number of points at which computations are performed will be higher. Both these factors will reduce the effect of vectorization overhead.

Appendix B Algorithm for calculation of IR fluxes.

The calculation of the IR fluxes at any level in either direction is fairly straightforward once the adjusted source function is defined. We describe first our strategy for the upward flux. The same strategy turned upside down is then applied to the downward flux. The basic idea is to sweep through the levels from bottom to top, adjusting the source function as necessary and calculating the flux at the current level as the sum of the contributions from all lower levels.

Initially the source function is set to the clear column source function. As we progress up the column, no adjustment is needed until we enter a cloud. Whenever there is a cloud in the layer directly below the current level we adjust the source function below the current level and at the boundary using the temperature of the current level.

It is instructive to consider what happens for levels within a cloud where there should be not net flux. When we calculate the downward fluxes, the algorithm will detect a cloud just above the current level and we will again use the temperature of the current level to adjust the source function at and above the current level. Thus in our calculation the contribution from the fraction of the sky with cloud at the current level will use a constant adjusted source function and yield no net flux.

Once we have passed through a cloud, further adjustments to the source function are not needed, until another cloud is encountered, because the adjustments have already been made for the cloud below the current level.

With the approach we have described, at level *k*, we need adjust the source function at the boundary and in layers *1* through *k* only for those points with cloud in layer *k*. To describe the algorithm, let B and B_ be the clear column source function in the layers and in the levels. Let C and C_ be the corresponding adjusted cloudy source function, averaged over the cloud formations, in this case only the clear and cloudy formation, where *f* is the cloud fraction. (For C_, only C_(0) or C_(Nk) is used.)

We can eliminate storing the transmittances (and C_) as follows. If we define C(0)=C_(0), and note that the zero path transmittance T(k,k) = 1, then the sum for the flux

$$\mathrm{F(k)} = \mathrm{C\_(0)*T(0,k)} + \sum_{j=1}^{k} \mathrm{(C(j)*(T(j,k)-T(j-1,k)))}$$

can be written as

$$\mathrm{F(k)} = \sum_{j=0}^{k-1} \mathrm{(C(j)-C(j+1)*T(j,k))} + \mathrm{C(k)}.$$

It is also convenient to define B(0)=B_(0), so the boundary term can be adjusted along with the atmospheric source function.

```
A sketch of the algorithm for the upward flux is:

   Initialize C = B, F(0) = C(0)
   For k = 1, Nk
```

```
      If there is a cloud in layer k then
         For j = 0, k
            C(j) = f*B_(k) + (1-f)*B(j)
         Endfor j
      Endif

      For each gas
         F(k)=C(k)
      End for each gas
      For j = 0, k
         For each gas
            Calculate T the transmission from
               level j to level k, which
               depends on theta(j)
            F(k) = F(k) + T*(C(j)-C(j+1)
         End for each gas
      Endfor j

   Endfor k
End of the upward flux calculation
```

The procedure for the downward flux is similar, but turned on end. Note that B_(Nk) = 0. Also level *k* is the boundary on layer *k+1*. Here we define C(Nk+1) = C_(Nk) and B(Nk+1) = B_(Nk) = 0.

```
For the downward flux,

   Initialize C = B, F(Nk) = C_(Nk)
   For k = Nk-1, 0, -1
      If there is a cloud in layer k+1 then
         For j = k+1, Nk+1
            C(j) = f*B_(k) + (1-f)*B(j)
         Endfor j
      Endif

      For each gas
         F(k)=C(k+1)
      End for each gas
      For j = k+1,Nk
         For each gas
            Calculate T the transmission from
               level j to level k, which
                  depends on theta(j)
               F(k) = F(k) + T*(C(j+1)-C(j))
            End for each gas
         Endfor j

      Endfor k
   End of the downward flux calculation
```

For vectorization each of the inner most statements is a loop on longitudes. The IF-block which adjusts the source function should probably not be vectorized. For each *k* it is active only for the fraction of points with clouds at layer *k*. This section is active only for a range of *k* (4 to 12 for the 18 layer model). It could be vectorized with gather/scatters or by double subscripting and creating a list of cloudy points.

Appendix C Solution for the diffuse solar flux

In the main text, by applying the adding method, we obtained the following equations for the diffuse flux leaving layer k:

$$\overline{G}^{\uparrow}_{k-1} = r_k G^{\downarrow}_{k-1} + \bar{r}_k \overline{G}^{\downarrow}_{k-1} + \bar{t}_k \overline{G}^{\uparrow}_{k} \tag{20}$$

$$\overline{G}^{\downarrow}_{k} = \bar{t}_k \overline{G}^{\downarrow}_{k-1} + \bar{r}_k \overline{G}^{\uparrow}_{k}. \tag{21}$$

The boundary conditions are

$$\overline{G}^{\downarrow}_{0} = 0 \tag{22}$$

$$\overline{G}^{\uparrow}_{N} = r_s G^{\downarrow}_{N} + \bar{r}_s \overline{G}^{\downarrow}_{N}. \tag{23}$$

If we write these equations in matrix form, we obtain a pentadiagonal system. However the special form of the system allows us to solve it with only two sweeps, in a fashion akin to the method normally used to solve tridiagonal systems. The approach is to solve each pair of equations in terms of the $\overline{G}^{\uparrow}_{k}$, the diffuse flux entering the bottom of the layer. In the next layer $\overline{G}^{\uparrow}_{k}$ is determined in terms of $\overline{G}^{\uparrow}_{k+1}$ and so on. In the last layer we obtain an equation relating the diffuse fluxes at the surface which together with (23) allows us to solve for $\overline{G}^{\uparrow}_{N}$. Then a straightforward backsubstitution yields all the fluxes.

To show that this approach works and to determine the algorithm for solving the system, we assume that in layer k we have determined a_k, b_k, c_k and d_k, such that

$$\overline{G}^{\uparrow}_{k-1} = a_k + b_k \overline{G}^{\uparrow}_{k} \tag{24}$$

$$\overline{G}^{\downarrow}_{k} = c_k + d_k \overline{G}^{\uparrow}_{k}. \tag{25}$$

If we specialize (20) and (21) for $k = 1$, we obtain

$$\overline{G}^{\uparrow}_{0} = r_1 + \bar{t}_1 \overline{G}^{\uparrow}_{1} \tag{26}$$

$$\overline{G}^{\downarrow}_{1} = \bar{r}_1 \overline{G}^{\uparrow}_{1}, \tag{27}$$

so that $a_1 = r_1$, $b_1 = \bar{t}_1$, $c_1 = 0$ and $d_1 = \bar{r}_1$. Now assuming that (24) and (25) hold for layer $k = n$ and that (20) and (21) hold for layer $k = n + 1$, then we find by substituting (25) into (20) that (24) holds for $k = n + 1$ provided

$$a_{k+1} = \left(r_{k+1} G^{\downarrow}_{k} + \bar{r}_{k+1} c_k\right)/(1 - \bar{r}_{k+1} d_k) \tag{28}$$

$$b_{k+1} = \bar{t}_{k+1}/(1 - \bar{r}_{k+1} d_k) \tag{29}$$

This result may be used in (25) with $k = n$ to determine $\overline{G}^{\downarrow}_{n}$ which we substitute into (21) with $k = n + 1$, finding that (25) holds for $k = n + 1$ provided

$$c_{k+1} = \bar{t}_{k+1}(c_k + d_k a_{k+1}) \tag{30}$$

$$d_{k+1} = \bar{r}_{k+1} + \bar{t}_{k+1} d_k b_{k+1}. \tag{31}$$

Thus we may easily determine a_k, b_k, c_k and d_k for $k = 1, \ldots N$. Equation (25) with $k = N$ substituted into (23) yields

$$\overline{G}_N^{\uparrow} = \left(r_s G_N^{\downarrow} + \bar{r}_s c_N\right)/(1 - \bar{r}_s d_N). \tag{32}$$

Then for $k = N, \ldots 1$ we may evaluate (24) and (25) for all the fluxes.

The solution just determined is always possible to obtain. The only potential difficulty is if the divisor in (28), (29) or (32) becomes zero. In fact, $b_k \geq 0$ and $d_k < 1$ for all k provided the values of $\bar{t}_k$ and $\bar{r}_k$ are reasonable. We now prove this assertion, assuming that

$$\begin{gathered} 0 \leq \bar{t}_k \leq 1 \\ 0 \leq \bar{r}_k < 1 \\ 0 \leq \bar{t}_k + \bar{r}_k \leq 1. \end{gathered} \tag{33}$$

(We do not allow $\bar{r}_k = 1$ because the case of two perfect reflectors facing each other is indeterminate and we should not expect to obtain a unique solution.) First, the assertion holds for $k = 1$, since $b_1 = \bar{t}_1$ and $d_1 = \bar{r}_1$. Now assuming the assertion holds for layer k, since $d_k < 1$ and (33), the divisor $(1 - \bar{r}_{k+1} d_k) > 0$. If $\bar{t}_{k+1} = 0$, then $b_{k+1} = 0$ and $d_{k+1} = \bar{r}_{k+1}$ and the assertion holds. If $\bar{t}_{k+1} > 0$, then $b_{k+1} > 0$ and we assume $d_{k+1} \geq 1$ to obtain a contradiction, thereby proving the assertion by induction. Assuming $d_{k+1} \geq 1$ and $\bar{t}_{k+1} > 0$, and making use of (31) and (33) leads to $d_k \geq 1/b_{k+1}$. Then (29) yields $d_k\left(\bar{t}_{k+1} + \bar{r}_{k+1}\right) \geq 1$, which is impossible according to (33) if $d_k < 1$. This contradiction proves $d_{k+1} < 1$.